UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche Fisiche e Naturali

Dipartimento di Tecnologie dell'Informazione

Un sistema di sicurezza per ambienti peer-to-peer

RELATORE
Prof. Pierangela Samarati

CORRELATORI
Prof. Ernesto Damiani
Dott. Sabrina De Capitani di Vimercati

Tesi di Laurea di:
Fabrizio CORNELLI
Matricola 587183

Anno Accademico 2001/2002

Prefazione

Non c'è nessun giusto, neppure uno solo. Tutti hanno traviato, tutti si sono corrotti, non c'e' nessuno che faccia il bene, neppure uno.

Romani 3.10-12

Sta sempre più aumentando l'utilizzo di Internet come mezzo per la condivisione e lo scambio di informazioni e risorse. Testimonia questa tendenza l'interesse crescente per i sistemi *peer-to-peer*, quali ad esempio Gnutella, da parte della comunità scientifica ed industriale.

Tali sistemi, nei quali ogni nodo, denominato *servent*, svolge mansioni sia di *server* sia di *client*, sono caratterizzati da un lato da requisiti di privatezza e dall'altro di protezione. Da un lato i partecipanti, infatti, non vogliono che la loro attività e le loro transazioni siano registrabili. Dall'altro gli stessi vogliono garanzie sulla qualità e l'integrità delle risorse disponibili in rete. Esiste infatti la concreta possibilità che questi sistemi vengano utilizzati per scopi illeciti e che possano favorire il dilagare di Virus e Trojan Horse.

Nei sistemi dove sia possibile affidarsi al giudizio super partes di qualche ente, il problema viene risolto con le *Certification Authority*, che si fanno carico di garantire la veridicità delle informazioni legate ai server. PGP, d'altro canto, offre un esempio di gestione distribuita della fiducia, che tuttavia non può essere utilizzata in ambienti anonimi.

Scopo della tesi è lo sviluppo di un sistema di sicurezza per sistemi peer-to-peer che permetta il controllo e la regolamentazione delle transazioni mediante l'introduzione di un metodo per valutare la *fiducia* (*reputazione*) che viene posta dai vari partecipanti nei loro pari, mantenendo al contempo l'anonimato delle transazioni e la completa decentralizzazione.

Il progetto di tesi si è sviluppato in più fasi e i contributi si articolano come segue:

- *Descrizione del contesto e del problema:* Sono state analizzate tutte le più importanti tecnologie peer to peer per la condivisione di risorse e sono stati valutati i metodi e i protocolli per affrontare il problema di codificare e gestire le reputazioni in modo semi automatico.
- *Definizione del protocollo:* Valutata la necessità di definire un nuovo protocollo è stato scelto di estendere Gnutella, una piattaforma Open Source in continuo sviluppo. Questa permette di condividere risorse e di effettuare delle ricerche che vengono tradotte con messaggi nella rete. Il possessore della risorsa cercata risponde specificando come connettersi per trasferire il file. A questo punto, nel protocollo originale sarebbe stato necessario operare una selezione delle risposte sulla base di pochi elementi, nessuno dei quali in grado di offrire

garanzie sull'integrità dei file. L'estensione al protocollo, proposta nella tesi, permette al nodo interessato alla risorsa di chiedere ad altri nodi il loro giudizio sugli offerenti. Sulla base dei voti ricevuti il nodo potrà quindi valutare quale offerente può essere ritenuto più affidabile e dal quale quindi scaricare la risorsa cercata. Le votazioni, che vengono crittate per garantire la privatezza necessaria, servono sia per definire la reputazione dei nodi sia per valutare la credibilità dei votanti.

- *Analisi del protocollo:* Sono stati analizzati diversi tipi di attacchi al sistema che possono essere perpetrati in ambienti peer-to-peer e valutato come la soluzione proposta possa controllarli.

- *Implementazione:* Viene analizzata la struttura dell'implementazione che tende a minimizzare l'impatto sulla comunità incapsulando i nuovi messaggi necessari al funzionamento del protocollo in quelli esistenti, così da permettere l'interoperabilità del prototipo con le implementazioni esistenti.

Gnutella è molto studiata e in continua evoluzione. Una delle tendenze più promettenti introduce i supernodi nella rete, strategia che offrirebbe maggior scalabilità accentuando tuttavia alcuni problemi già noti. È previsto uno sviluppo di questo lavoro di tesi per trattare la nuova topologia, sfruttando i nodi privilegiati non solo per gestire in modo più efficiente le ricerche, ma anche per convogliare in modo sicuro le reputazioni.

La tesi è organizzata come segue.

Nel *Capitolo 1*, dedicato all'introduzione della tesi, si definiscono i principali concetti degli ambienti peer-to-peer e le loro possibili vulnerabilità dal punto di vista della sicurezza. Inoltre si definisce il concetto di reputazione in ambito informatico e si mostrano alcuni protocolli che l'utilizzano.

Nel *Capitolo 2* viene presentata una tassonomia delle reti peer-to-peer evidenziando le caratteristiche delle varie topologie, allo scopo di meglio comprendere l'analisi successiva di tutte le tecnologie attualmente disponibili. Alcune di queste, come ad esempio Napster, vengono inquadrate nella situazione socio-politica di Internet, dal momento che sono uscite dall'ambito prettamente tecnico per invadere, con loro conseguenze sociali ed economiche, i media di tutto il mondo. Viene quindi analizzato il protocollo Gnutella, che verrà esteso nel lavoro di tesi.

Nel *Capitolo 3* vengono illustrati i problemi legati a Gnutella e viene proposto un protocollo per l'introduzione del concetto di reputazione.

Nel *Capitolo 4* viene descritta l'implementazione del protocollo proposto nel lavoro di tesi.

Nel *Capitolo 5* vengono presentate le conclusioni con alcuni commenti sui protocolli trattati e si descrivono alcuni possibili sviluppi futuri.

Ringraziamenti

Vorrei ringraziare tutti quanti mi hanno aiutato. In particolare:

La relatrice, Prof.ssa Pierangela Samarati, e i correlatori, Prof. Ernesto Damiani e la Dott.ssa Sabrina De Capitani di Vimercati.

Lytcha, i miei genitori, mia sorella e la Dott.ssa Bisson. Luigi e Costanza.

Mirco e Alberto, Barbara, Domenico, Cubo, Luca, Serena, Mario e Marcello, Mauro, Olga e Sandro, Ben, Pablo, Fabio, Colombo, Sbra, Rondo, Cariù, Tapongo, Tzu, Enzo, Valerio, tutto il gruppo Malawi, il laboratorio Turing, la sala macchine, il Linux User Group di Crema, LoLug.

Andrea, Paolo e Fabi, Luca, Pier, Giamba.

Tutti i docenti del Polo di Crema, e in particolare il Prof. Fornili, il Prof. Righini, il Prof. Fiorentini, il Prof. Degli Antoni e il Prof. Scarabottolo.

Jan Anderson.

Indice

Capitolo 1

Introduzione

Ma una notizia un po' originale
non ha bisogno di alcun giornale
come una freccia dall'arco scocca
vola veloce di bocca in bocca.

Fabrizio De André

Nel mondo delle tecnologie Internet, da qualche tempo, gli ambienti peer-to-peer stanno riscontrando notevoli successi, sia presso il grande pubblico, sia negli ambienti scientifici e accademici, sia nelle realtà industriali. Gran parte della fama è dovuta a quello che ormai comunemente viene definito *fenomeno Napster* [50] che ha coinvolto milioni di utenti e case discografiche di tutto il mondo. In contemporanea, e grazie allo slancio ottenuto durante il periodo di crescita di quel prodotto, sono nati diversi altri progetti con idee di base anche profondamente diverse. Napster è una rete di utenti (peer) che si connettono ad un server centrale e condividono delle risorse (file musicali). Il server si occupa dell'indicizzazione dei contenuti dei repository degli utenti e fornisce un metodo per mettere in contatto diretto (*peer-to-peer*) gli utenti che vogliano scambiarsi materiale. Fu proprio il fatto di essere un meccanismo centralizzato a fornire alle case discografiche e alle istituzioni la leva per forzare e bloccare il sistema; effettivamente molto del materiale scambiato privatamente sottostava a limitazioni di copyright. Attualmente gli ambienti peer-to-peer sono sotto continuo attacco da parte di enti come RIAA (Recording Industry Association of America) e MPAA (Motion Picture Association of America) che vogliono proteggere i diritti intellettuali sul materiale che viene comunemente scambiato senza controllo.

Naturalmente non si intende con questo lavoro favorire lo scambio sregolato di materiale sotto copyright, bensì di fornire dei meccanismi per riuscire a valutare la moralità, l'integrità e, se possibile, anche la qualità delle risorse e dei server che le distribuiscono.

I metodi utilizzati nelle reti peer-to-peer trattati in questa tesi si focalizzeranno su sistemi decentralizzati e in particolare si tratterà con attenzione *Gnutella* [73] (per il quale si proporranno delle estensioni al protocollo), *Freenet* [16], come esempio di ambiente anonimo, *KaZaA* [39] per il fatto di essere l'implementazione commerciale peer-to-peer tecnologicamente più avanzata.

Ad una trattazione generale degli ambienti peer-to-peer seguirà nel capitolo la descrizione dei metodi utilizzati per includere il concetto di reputazione nelle reti di Internet, distinguendo gli approcci centralizzati da quelli decentralizzati.

1.1 Peer to Peer

Il termine peer-to-peer è un termine generico per rappresentare tutte le architetture nelle quali tutti i nodi offrono gli stessi servizi e seguono lo stesso comportamento. La grande differenza di un ambiente di questo tipo rispetto ad una semplice navigazione Web sta nella partecipazione più intensa dell'utente, che esce dal ruolo di mero spettatore di una realtà preconfigurata e condivide le proprie risorse per creare una nuova rete eterogenea, imprevedibile e difficilmente controllabile [14]. La struttura appare essere un client-server dinamico, nella quale i ruoli non sono definiti a priori, ma cambiano a seconda delle necessità; pertanto i nodi di una rete siffatta vengono chiamati *servent*. Questa tecnologia riesce ad essere una valida infrastruttura in grado di far condividere risorse, informazioni, potenza di calcolo e spazio tra sistemi eterogenei.

Uno dei vantaggi fondamentali delle architetture peer-to-peer sta nel fatto di garantire un'eccellente scalabilità, essendo possibile arrivare ad avere centinaia di migliaia di nodi interconnessi a costi molto bassi. Del resto uno dei sistemi che ha garantito a costi bassissimi un'enorme capacità di calcolo fonda il suo principio proprio sull'utilizzo contemporaneo di migliaia di workstation sparse per la rete: *SETI@home* [68]. SETI è un progetto per la rilevazione di sorgenti extraterrestri il cui metodo di ricerca consiste nell'analizzare lo spettro rilevato da radiosorgenti con strumenti matematici basati su trasformate di Fourier, note per essere molto impegnative dal punto di vista computazionale. Eventuali "anomalie" vengono riscontrate e successivamente analizzate, nella speranza di rilevare eventuali comunicazioni di intelligenze extraterrestri. Essendo il problema facilmente parallelizzabile, è stato proposto alla comunità Internet internazionale un software gratuito, scaricabile dal sito del progetto, in grado di connettersi al loro database, di scaricare le specifiche per portare a termine una parte di lavoro, eseguire i calcoli e restituire il risultato, con un consumo minimo di banda e l'utilizzo della CPU nei momenti nei quali giacerebbe inutilizzata. Questa idea ha riscosso un notevole successo e migliaia di persone in tutto il mondo hanno cominciato a competere per riuscire a elaborare il maggior numero di pacchetti possibile, gratificati dal vedere il proprio nome salire in una classifica pubblica. Non si può propriamente parlare di peer-to-peer per SETI perché si tratta di una architettura distribuita. Tuttavia ciò che pare importante evidenziare è invece che il potenziale di calcolo e di memorizzazione composto da tutte le macchine connesse in rete è enorme e allettante, per moltissime applicazioni. [1]

Tuttavia, al crescere della popolazione, e con l'introduzione di servizi che mettono in contatto diretto nodi diversi, la fiducia reciproca tra utenti, che in piccole comunità spesso non presenta problemi, diventa una questione della massima importanza, essendo in qualche sua forma, spesso indispensabile alla struttura stessa dei servizi. Si pensi, ad esempio, a siti che propongono aste elettroniche o anche siti nei quali vengono presentati i commenti a merce in commercio. In questi casi riuscire a distinguere le persone oneste da quelle che agiscono con scopi antisociali diventa assolutamente fondamentale.

A seconda degli obiettivi che si vogliono raggiungere, può esserci la necessità di garantire l'anonimato di chi effettua le transazioni. Diversi lavori hanno proposto di organizzare le reti in gruppi di nodi che sacrificano banda e risorse per ottenere anonimato: alcuni esempi sono APFS [75], Onion Routing [42], Hordes [13], Crowds [41] e FreeNet [16]. In generale, queste proposte prevedono che, per effettuare una connessione, vengano interpellati diversi altri nodi che tramite tecniche crittografiche, non hanno modo di sapere nè il contenuto dei messaggi, nè le parti coinvolte nella comunicazione.

[1] Alcune valutazioni [17] dichiarano che nel mondo, a Marzo 2002, ci sono 170.849.000 host

Garantire l'anonimato in senso stretto non è sempre necessario. In molti casi è sufficiente identificare gli utenti con pseudonimi, mascherando il loro nome pur lasciando visibile il loro indirizzo IP di provenienza. Si preferisce utilizzare lo pseudoanonimato sull'anonimato stretto quando l'efficienza è considerata più importante della sicurezza.

Indipendentemente dal tipo di anonimato utilizzato nelle reti, è molto spesso importante riuscire ad attribuire alle azioni degli utenti un valore di qualità. Per ottenerne una misurazione obiettiva serve che questa rifletta la reputazione acquisita dall'utente, in modo da far risaltare le azioni di coloro che hanno saputo garantire a lungo un buon comportamento. In questi casi è necessario stabilire una metrica per misurare e gestire la reputazione. Sono state presentate diverse soluzioni di cui vengono tracciate le linee più importanti.

1.2 Gestire la reputazione

Molte comunità virtuali hanno la necessità di effettuare transazioni per le quali è necessaria la reciproca fiducia dei partecipanti, sentimento naturale nei rapporti umani, funzione di precognizione statistica piuttosto difficile da definirsi e da gestire in ambienti informatici. La fiducia infatti, strettamente connessa alla reputazione è stata definita come segue [20]:

La fiducia (e simmetricamente la sfiducia) è un livello soggettivo particolare della probabilità con la quale un individuo effettuerà una particolare azione, prima che sia verificabile e in un contesto nel quale possa influire sulle nostre azioni.

Matematicamente, la fiducia ha certe proprietà che la rendono inadatta ad essere considerata una metrica (ad esempio non vale la proprietà transitiva). Pertanto si tende a discutere in modo rigoroso quella che viene definita "probabilità soggettiva", per indicare l'esistenza di diversi livelli di fiducia.

Quando si devono prendere delle decisioni, ogni individuo deve valutare tutti gli aspetti di una particolare situazione, spesso nelle condizioni di avere soltanto un'informazione parziale; del resto se fosse totale non sarebbe necessario introdurre il concetto di fiducia, poiché tutto sarebbe noto. Nella nostra società otteniamo queste informazioni chiedendo ad altre persone di fiducia un giudizio sulla reputazione di colui sul quale stiamo indagando. Ogni volta che si ottengono informazioni di questo genere si tende a modificare non soltanto la considerazione riguardante la persona sotto giudizio, ma la credibilità di chi lo esprime. Questo comportamento, complesso argomento sociologico, è una forma di controllo sociale [3] che mantiene unite le società, dove il comportamento di ogni individuo è condizionato da quello degli altri, e dove la maggior parte delle persone, per indole umana, agiscono in modo cooperativo. Ad esempio: un negoziante disonesto guadagnerebbe molto in fretta una cattiva reputazione nel vicinato, e alla lunga dovrebbe chiudere o cambiare politica. La buona reputazione, invece, deve essere vista come una sorta di investimento a lungo termine, una forma di capitale sociale, indispensabile soprattutto nel commercio.

La reputazione è stata così definita [3]:

La reputazione è l'aspettativa sul comportamento di un individuo sulla base delle informazioni raccolte sul suo precedente comportamento.

La metrica di reputazione, ovvero la proiezione matematica di questo concetto, così naturale da sconfinare nell'irrazionale, è tipicamente una raccolta di opinioni di utenti che hanno avuto rapporti d'affari con l'oggetto, la persona o il servizio in questione: ogni utente indicherebbe se è

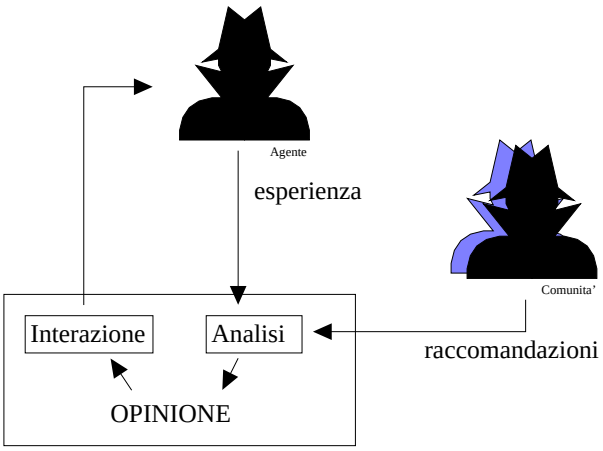


Figura 1.1: Modello generico di fiducia

stato soddisfatto o meno. Nel caso più semplice, la reputazione è la valutazione media ricevuta da tutti gli utenti che hanno interagito con esso, ma ogni sistema tende ad adattare la metrica alle proprie esigenze. Esistono infatti diverse metriche [3, 4, 54] più raffinate che permettono di dare definizioni matematiche formali alla reputazione e alla fiducia, in modo da poterle trattare sistematicamente. Un approccio sistematico è indispensabile, ad esempio, nel commercio elettronico, nuovo motore economico di Internet. Ad esempio, il Trust Model [3], prevede che la fiducia sia soggetta alle seguenti proprietà:

- La fiducia dipende dal contesto.
- Ci sono quattro valori che rappresentano i gradi positivi e negativi di fiducia.
- La fiducia si basa sull'esperienza passata.
- Gli individui sono in grado di scambiarsi informazioni riguardanti la fiducia sottoforma di raccomandazioni.
- La fiducia non è transitiva: le raccomandazioni non vengono necessariamente tradotte in fiducia.
- La fiducia è soggettiva: osservatori diversi possono avere percezioni diverse.
- La fiducia è dinamica e non monotona, può aumentare o diminuire.

Un'astrazione generica del modello Trust Model, valida per tutti i modelli che fanno uso della reputazione, è illustrato in Figura 1.1. Esso consiste di due attori: l'*individuo* e la *comunità*. La fiducia è definita dal risultato della valutazione delle esperienze dirette unite alle raccomandazioni della comunità, alcune delle quali accettabili, altre tendenziose o faziose e pertanto da riconoscere.

Ogni particolare modello specifica in modo differente come queste proprietà debbano essere garantite, quali valori attribuire alla scala della fiducia e come aggiornare i parametri di valutazione per mantenere dinamico il sistema.

Nell'ambito della nostra ricerca, i modelli per la gestione della fiducia, possono essere distinti in due categorie: quelli ad *approccio centralizzato* e quelli ad *approccio distribuito*. Mentre il primo prevede una struttura centralizzata con un'autorità di fiducia universale, il secondo assume che ogni individuo abbia la libertà di scegliere di chi fidarsi, ottenendo spesso delle soluzioni più difficili da gestire.

1.2.1 Approcci centralizzati

Un gestore di reputazione ad approccio centralizzato è un servizio indipendente che si tiene al corrente della qualità, della credibilità e della reputazione di ogni elemento in un insieme. Tipicamente si ha la necessità di valutare siti web, aziende, prodotti o anche persone. Più genericamente qualunque cosa che possa essere osservata prima di effettuare delle transazioni che la riguardino è soggetta a reputazione.

Diverse esigenze hanno prodotto modelli per la gestione della reputazione; nel seguito vengono presentate alcuni dei principali sistemi commerciali ad approccio centralizzato che fanno uso di reputazione.

eBay

eBay [23] è un'asta elettronica dove ogni utente registrato che desideri vendere un prodotto, può indire un'asta specificando i dettagli del prodotto e il prezzo base dal quale partire. Chi sia interessato all'acquisto, dopo essersi anch'egli registrato al servizio, ha modo di proporre un prezzo e, nel caso in cui vinca l'asta, ha il diritto (e dovere) d'acquisto. I pagamenti avvengono in modo diretto o tramite un servizio messo a disposizione di eBay stesso, chiamato *ePay*, che come una banca elettronica, permette di trasferire denaro in modo semplice, tramite carta di credito. eBay è stato uno dei primi siti a tener traccia della reputazione dei vari utenti che partecipano al servizio. Dal momento che le transazioni avvengono tra sconosciuti spesso di diversi paesi, è infatti necessario che ci sia un metodo per valutare la credibilità dei partecipanti. Che fare infatti, se chi vende non spedisce effettivamente la merce? Che fare se chi compra non paga?

Ad ogni acquisto, il compratore fornisce a eBay un giudizio esteso e uno sintetico sul venditore. Il giudizio sintetico può assumere i valori di buono, medio e scarso, mentre il giudizio esteso è una frase che ne spieghi le ragioni. Possibili compratori si possono sentire tranquillizzati dal riscontrare che le recenti transazioni del venditore sono terminate con la soddisfazione dei clienti precedenti. D'altra parte i venditori sono incentivati a mantenere un buon comportamento da questo sistema, poichè basta una sola cattiva esperienza per rovinare la reputazione di un venditore.

eBay ha quindi un gestore di reputazione centralizzato con utenti registrati non anonimi, cosa che implica il fatto di non poter abbandonare l'account nemmeno in caso di pessima reputazione. Questo meccanismo è supportato inoltre da un'assicurazione che copre parte dei danni dovuti ad un suo eventuale cattivo funzionamento.

Epinions

Epinions [27] permette di esprimere opinioni su una serie vastissima di prodotti e di servizi in vendita in rete, dai computer palmari ai musei di NewYork. Chi abbia necessità di acquistare un prodotto in Internet si trova spesso nella condizione di non sapere di chi fidarsi e nel timore di spendere più denaro del necessario e di non acquistare la scelta migliore. Epinions, grazie ai contributi degli utenti, riesce ad attribuire una reputazione ai vari prodotti in vendita e permette di confrontarne le caratteristiche, così da permettere di valutare la scelta migliore. Scelto il prodotto è possibile continuare la navigazione controllando le valutazioni dei siti che lo vendono, trovando, inoltre, un confronto sui prezzi. Questo sito ha introdotto per la prima volta il concetto di credibilità associato agli utenti che scrivono i commenti ai prodotti, in modo da ridurre o evitare del tutto commenti faziosi. Ogni commento può essere giudicato utile o meno da parte

degli altri lettori, giudizio che serve a modificare la *credibilità* di chi lo ha scritto. Gli utenti che hanno acquisito una credibilità molto alta hanno maggior peso nella valutazione complessiva e hanno diritto a maggior visibilità nell'elenco dei giudizi estesi. Questo modello, chiamato *gestione a doppia reputazione*, è diventato attualmente lo standard de facto, dimostratosi infatti un'ottima soluzione.

Amazon

Amazon.com [5], una delle aziende che per prima ha sfruttato il commercio elettronico diventando un riferimento mondiale per la vendita on-line di libri, ha da subito incluso la possibilità di lasciare ai lettori la libertà di aggiungere commenti ai libri in vendita. Purtroppo questa idea, che poteva apparire ingenuamente un modo per migliorare il servizio e per coinvolgere gli utenti, si è rivelata essere un'arma molto potente utilizzabile contro autori e libri. Infatti è capitato che alcuni libri e alcuni autori venissero presi di mira da loro detrattori ed insultati pubblicamente, con devastanti conseguenze sulle vendite. A questi commenti seguivano tipicamente elogi e panegirici da parte di amici, mandando in confusione i compratori. Ora vengono effettuati dei controlli ed è stata introdotta la doppia reputazione.

Google

Google [32] è un motore di ricerca tra i più efficienti ed innovativi: restituisce la lista delle pagine che soddisfano la ricerca sulla base di un valore di qualità che dipende dal numero di documenti che si riferiscono alla pagina stessa. Questa soluzione, difficilmente estendibile ad altri ambienti, è stata sfruttata da alcune aziende che sono riuscite ad aumentare la propria reputazione, e quindi la probabilità di comparire nelle ricerche, creando migliaia di pagine piene di riferimenti ai prodotti da pubblicizzare.

Slashdot

Slashdot [67], forum di discussione, permette agli utenti di definire, per ogni commento, un giudizio. In modo da permettere la visione selettiva delle migliaia di mail che vengono pubblicate quotidianamente. Ad ogni utente registrato viene associata una reputazione, denominata *karma*, che viene modificata sulla base dei giudizi espressi dai lettori sui suoi messaggi. Un buon comportamento innalza il *karma*, e i messaggi avranno maggior peso, e pertanto avranno maggior probabilità di essere effettivamente letti.

1.2.2 Approcci distribuiti

Tutti gli approcci visti precedentemente richiedono la presenza di un server centralizzato che possa associare ad ogni utente un valore di reputazione, univoco e super partes. Queste soluzioni, per quanto interessanti, non sono applicabili ad ambienti distribuiti, come ad esempio le reti peer-to-peer. Esistono poche soluzioni proposte per ambienti distribuiti e solo poche di queste effettivamente implementate. Segue una descrizione di PGP che implementa *Web of Trust*[1], un modello di reputazione storico da cui nessun'altro nuovo modello può prescindere. Successivamente vengono descritti *Poblano*, *Global Trust* e *Free Heaven*.

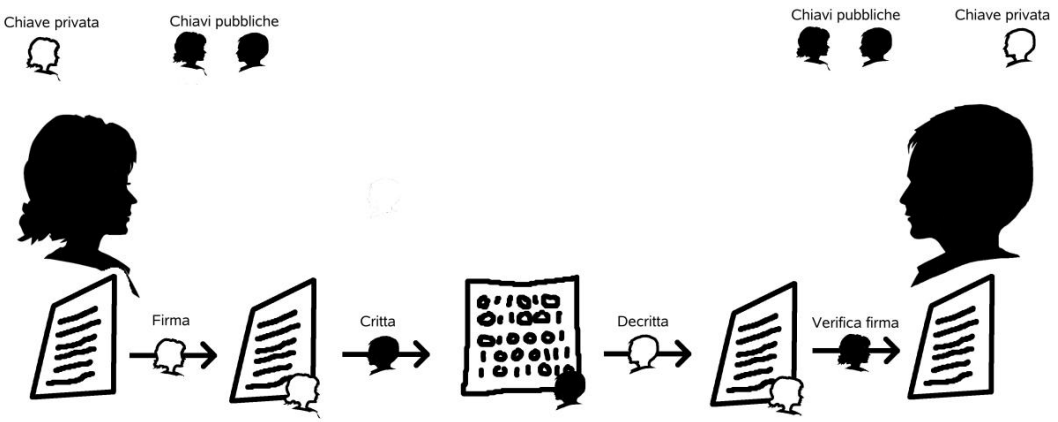


Figura 1.2: Firma digitale e crittografia asimmetrica

PGP

Un approccio decentralizzato storico è stato implementato in *Pretty Good Privacy* [1, 2, 80] un sistema crittografico a chiavi asimmetriche. *PGP* è una pietra miliare nella storia dei sistemi crittografici perché ha permesso alle grandi masse che popolano Internet di utilizzare per la prima volta gratuitamente e con semplicità la crittografia di grado militare. *PGP* è nato principalmente per crittare e-mail usando chiavi asimmetriche ma supporta anche la crittografia simmetrica, principalmente per la crittazione di file locali. La crittografia asimmetrica prevede che ci sia un mittente e un destinatario, dotati di una propria chiave pubblica, di una propria chiave privata e della chiave pubblica dell'altro. Il sistema prevede che per mandare un messaggio crittato ad un utente il messaggio di testo debba venire crittato usando la chiave pubblica del ricevente. Un'altra caratteristica estremamente interessante di questo software è che permette di firmare i messaggi in modo da garantire, oltre alla riservatezza, anche l'autenticità del mittente.

La Figura 1.2 illustra lo schema utilizzato per la firma digitale e per la crittazione: Alice, che vuole spedire un messaggio firmato e crittato a Bob, procede come segue:

1. Alice recupera la chiave pubblica di Bob.
2. Alice firma il messaggio con la propria chiave privata.
3. Alice critta il messaggio firmato con la chiave pubblica di Bob.
4. Alice spedisce il risultato dell'operazione precedente a Bob.
5. Bob riceve il messaggio e lo decritta con la propria chiave privata, l'unica in grado di decifrare il messaggio.
6. Bob verifica la firma digitale con la chiave pubblica di Alice.

Si nota da questo schema come il rapporto tra la chiave pubblica e privata sia stretto e al contempo complementare.

Quello che questo schema non dice è come Alice e Bob possano scambiarsi la chiave pubblica in modo sicuro. Infatti, se Alice spedisse la propria chiave pubblica a Bob in un canale insicuro, non darebbe a Bob la garanzia di ricevere la chiave esatta. Un possibile attacco chiamato *man in the middle* prevede infatti che un terzo partecipante si collochi nel mezzo della trasmissione, facendosi credere Alice a Bob, e viceversa.

Risultato di questa situazione, dovuta al fatto che né Bob né Alice hanno avuto modo di scambiarsi in modo sicuro la chiave pubblica, è che tutte le transazioni successive sono diventate a rischio. Si consideri che uno scenario di questo genere è ben lungi dall'essere inattuabile: spesso i messaggi crittati passano via mail attraverso un certo numero, più o meno fisso, di server SMTP. All'attaccante basterebbe quindi prendere possesso di uno di questi nodi e modificare le regole di trasferimento in modo da attuare le modifiche necessarie ai messaggi.

Se ci fosse un'autorità di cui tutti conoscono la chiave pubblica allora lo scambio delle chiavi sarebbe semplice, poichè ad Alice basterebbe crittare la propria chiave pubblica con quella pubblica dell'autorità (Certification Authority), in modo che essa soltanto possa decifrare il messaggio. La Certification Authority restituirebbe ad Alice un certificato che ne autentichi la chiave pubblica, che può essere esibito a suo fianco. PGP non adotta questa tecnica, essendo un'architettura distribuita: preferisce che l'autenticazione avvenga tramite catene di utenti. L'idea di fondo è che, supponendo che Alice abbia la certezza di aver ricevuto proprio la chiave pubblica di Bob, potrebbe lei stessa fornire un certificato firmato per quella chiave. A questo punto, un terzo amico di Alice, che di lei ha massima fiducia, potrebbe, verificando la firma digitale, riconoscere l'autenticità della chiave pubblica di Bob e trasferirne di conseguenza la fiducia.

In questo modo sono state create delle catene di migliaia di chiavi.

Alla creazione di un certificato che autentichi una chiave pubblica, è possibile scegliere il livello di fiducia da attribuire alla chiave. La ragione per cui è necessario introdurre diversi livelli è per dare delle limitazioni alla proprietà altrimenti transitiva della fiducia, molto pericolosa quando le catene si allungano a dismisura. Certe volte infatti, pur essendo necessario includere una chiave nel proprio *key ring*, non si può avere la certezza assoluta di aver ricevuto la chiave dell'individuo a cui si pensa appartenga. Seguono i quattro livelli ammessi, con una breve descrizione.

- *Non definito* : non ci sono dati per attribuire alcuna fiducia alla chiave.
- *Nessuna* : la chiave pubblica non deve essere considerata valida e i certificati prodotti da questa non devono essere presi in considerazione.
- *Marginale* : la chiave deve essere creduta quando introduce un'altra, ma deve restituire un certificato di fiducia marginale.
- *Completa* : massima fiducia nella chiave. Ogni chiave firmata da questa deve essere presa in considerazione.

Detto questo rimane il problema di stabilire quando una chiave, con diverse firme, può essere considerata valida o meno. Ovviamente se tutte le firme fossero di completa fiducia e di chiavi a loro volta certificate, il dubbio non si porrebbe. Accade spesso però che alcune firme siano di reputazione marginale. In questo caso vengono contate e sulla base del loro valore, in relazione ad alcune variabili configurabili dall'utente che riflettono il suo scetticismo, si decide quale valore di fiducia attribuire alla chiave.

Il risultato di questa operazione è un livello di fiducia associato ad ogni certificato, che prevede tre livelli:

- *Non definito* : non si sa se il certificato sia valido.
- *Marginale* : si è abbastanza sicuri che il certificato sia valido.

- *Completo* : si ha la certezza assoluta che la chiave appartenga all'utente che la dichiara propria.

Il meccanismo di gestione delle chiavi a fiducia marginale è però risultato troppo complesso e soggetto a variabili locali, quindi ne viene sconsigliato l'utilizzo, se non in casi di necessità. Le grandi comunità che adottano questo sistema, come ad esempio Debian [18], usano certificare le chiavi sempre con la massima fiducia e infatti prevedono che i possessori delle chiavi si incontrino personalmente mostrando le proprie credenziali (carta d'identità o passaporto).

Poblano

Poblano [54] è un modello di fiducia sviluppato da Sun Microsystem espressamente per ambienti distribuiti peer-to-peer in architetture *JXTA* [31]. Come accennato, questi ambienti non prevedono delle strutture centrali che possano fungere da autorità e il problema di non aver strumenti per valutare la fiducia di un partecipante alle transazioni è presente e pressante. L'idea, come nel modello *PGP*, è quella di creare una catena di nodi reciprocamente connessi da un legame di fiducia, definito come un valore compreso tra -1 e 5, con questi significati:

Valore	Significato
-1	Sfiducia
0	Ignora
1	Fiducia minima
2	Fiducia media
3	Fiducia buona
4	Massima fiducia

Nel modello viene specificato come viene trasferita la fiducia e come associare un valore ad un server, tuttavia Poblano non ha riscosso molte approvazioni da parte della comunità scientifica.

Global Trust Model

Il modello Global Trust [38] assume per semplicità che esistano due soli valori di fiducia e che i nodi, per ogni transazione, possano comportarsi correttamente oppure no. Un'altra assunzione importante è che il numero di nodi scorretti sia basso e che per un osservatore non sia possibile determinare quale di due parti sia disonesta in una transazione non andata a buon fine. Il metodo proposto utilizza delle griglie di routing, chiamate *P-Grid*, che permettono di gestire le comunicazioni dei nodi mantenendo la struttura scalabile e distribuita. Le operazioni ammesse in questa griglia sono la ricerca di chiavi e l'inserimento di valori associati a chiavi particolari. L'idea di base è di fare in modo che i valori di fiducia vengano inseriti in più copie (repliche) in questa struttura ed estratti per effettuare delle valutazioni statistiche sulla reputazione di qualche agente. Il modello non è mai stato implementato.

Free Heaven

Free Heaven [21] è un sistema di condivisione risorse anonimo e distribuito che cerca di garantire la libertà di espressione degli individui. Il modello adotta una definizione di fiducia, necessaria per stabilire delle connessioni sicure: ogni nodo infatti, afferma di garantire un certo comportamento, verificato nel tempo. La reputazione di ogni nodo viene così determinata dalla serietà

con la quale vengono mantenuti gli impegni presi, facilmente verificabili, essendo misurabili in termini di persistenza di file. Ogni nodo infatti deve dichiarare, nel momento in cui accetta di mantenere una chiave, il tempo per il quale è disposto a farlo. Questa chiave viene periodicamente controllata, per verificarne la presenza. Free Heaven non ha sviluppato una metrica particolarmente interessante né tantomeno portabile ad altri ambienti, essendo molto semplice verificare l'integrità di ogni nodo. Da questo fatto deriva che i meccanismi di propagazione della fiducia sono minimali. Del resto l'obiettivo del modello è di garantire persistenza dei file in modo anonimo e non si pone come risultato da perseguire la valutazione della fiducia di ogni nodo, essendo questa funzionale soltanto alla correttezza di un comportamento gestito internamente.

1.2.3 Commenti

Come si è visto esistono moltissime implementazioni di modelli per ambienti centralizzati, mentre per gli ambienti distribuiti ne esistono ben poche. Il modello Web of Trust di PGP, in grado di attribuire dei livelli di fiducia a dei certificati, ha il pregio di essere consilidato e ben studiato, ma non è facilmente estensibile ad ambienti dove la fiducia si fondi sull'esperienza, e sia di conseguenza una funzione del tempo e delle transizioni. Nemmeno Poblano e Free Heaven riescono a proporsi come modello generico, essendo troppo legati agli ambienti per i quali sono stati progettati. Persino Global Trust Model, pur essendo un'interessante trattazione teorica, offre pochi spunti per un'implementazione generica.

Gestire la reputazione in ambienti fortemente dinamici e pseudoanonimi è un problema al quale si cerca, nei prossimi capitoli, di trovare una soluzione.

Capitolo 2

Tecnologie peer-to-peer

I worry about my child and the Internet all the time, even though she's too young to have logged on yet. Here's what I worry about. I worry that 10 or 15 years from now, she will come to me and say "Daddy, where were you when they took freedom of the press away from the Internet?"

Mike Godwin, EFF

2.1 Ambienti P2P

Gli ambienti peer-to-peer non sono novità degli ultimi anni. Già trenta anni fa diverse compagnie lavoravano su architetture che ora sarebbero definite con questo nome [37, 53]. Ma in questi anni diversi fattori hanno favorito la crescita degli ambienti peer-to-peer: potenza di calcolo, banda e spazio a costi molto bassi. In contesti di questo genere, computer che tradizionalmente hanno coperto il ruolo di meri client, si trovano a comunicare direttamente tra loro, agendo contemporaneamente sia da server che da client, assumendo di volta in volta il ruolo che massimizzi l'efficienza della rete. Questa decentralizzazione riduce il carico sui server e permette loro di occuparsi di operazioni molto specializzate (come creazione di mailing-list, gestione dei pagamenti e così via) con maggior tranquillità. Inoltre permette alle aziende di espandersi senza la necessità di modificare in modo sostanziale l'infrastruttura della loro rete: un esempio chiarificatore è la gestione dei backup. In ambienti tradizionali, un servizio del genere richiederebbe un server di grosse dimensioni, piuttosto costoso. Lo stesso scopo può essere raggiunto connettendo in un cluster diversi computer di piccole o medie dimensioni in grado di condividere dello spazio in modo trasparente, efficiente e sicuro utilizzando tecniche peer-to-peer.

Le ragioni per le quali potrebbe essere utile l'uso di reti peer-to-peer sono molteplici ma attualmente vengono utilizzate in modo significativo quasi esclusivamente per *condividere* risorse software. Per accedere a queste risorse occorre trovarle e richiederle. La ricerca può avvenire in due modi: effettuando delle ricerche sul nome o sul contenuto, oppure in alcuni casi, scegliendole dalla lista delle risorse esportate da un particolare nodo di cui si conoscano le coordinate. Una

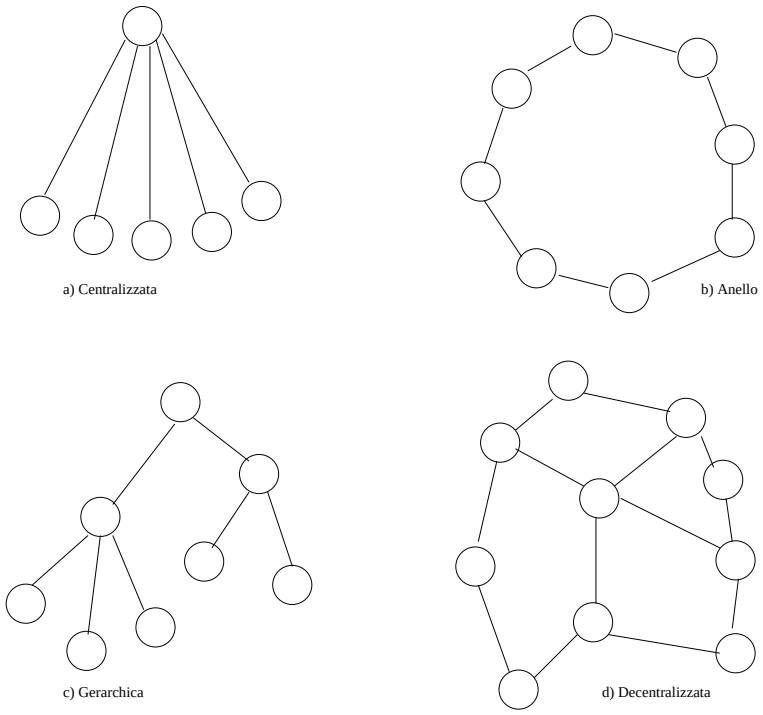


Figura 2.1: Topologie di reti P2P

volta scelta la risorsa da scaricare, avviene una connessione diretta tra i due servent che portano a termine la transazione, spesso indipendentemente dal resto della rete.

Le reti peer-to-peer si sono create uno spazio considerevole nella letteratura ma soprattutto sono di fatto divenute un metodo nuovo e molto popolare di scambiare risorse, legali e non. Le reti peer-to-peer sono tanto popolari da aver attirato l'interesse di milioni di utenti che hanno abusato delle libertà fornite e ne hanno fatto un libero scambio di software illegale. Questo comportamento diffuso ha attirato l'attenzione della stampa insieme alle ire di coloro che vorrebbero garantiti i diritti intellettuali del software scambiato, che naturalmente si sono sentititi defraudati dall'uso sconsiderato di questo mezzo; eclatante il caso Napster (vedi la sezione 2.3).

Per poter discutere in dettaglio delle caratteristiche tecniche di alcune di queste reti, occorre valutare l'impatto delle topologie di rete più spesso adottate negli ambienti peer-to-peer, che possono essere distinte in tipologie di base e topologie ibride.

2.1.1 Topologie di base

Tradizionalmente esistono diverse topologie di rete alcune delle quali sono state utilizzate per la costituzione di reti peer-to-peer [47]. In particolare, come illustrato in Figura 2.1, le topologie di base adottate in reti peer-to-peer possono essere: *centralizzate*, ad *anello*, *gerarchiche* e *decentralizzate*.

Centralizzata

In Figura 2.1(a) è raffigurata l'architettura tipica di una rete centralizzata: un server centralizzato riceve e gestisce le connessioni da parte dei client. Il server tipicamente risiede nell'azienda che offre il servizio e si occupa di mantenere in uno stato di integrità la struttura delle connessioni, fornendo a ciascun client informazioni inerenti alle risorse condivise

dagli altri. Ogni qual volta un client desideri effettuare delle ricerche all'interno della rete, manda un messaggio al server, il quale, avendo a disposizione gli indici delle risorse esportate da tutti i componenti della rete, estrarre i riferimenti alle risposte plausibili e le restituisce sotto forma di messaggio. Nelle reti peer-to-peer che utilizzano questo modello, lo scambio effettivo delle risorse non richiede l'intervento dei server, avvenendo in modo diretto tra i nodi della rete stessa.

Le reti centralizzate sono molto efficienti perché non c'è spreco di risorse nella gestione dell'infrastruttura e i protocolli, facendo riferimento ad un unico server autorizzato e autenticato, possono essere molto leggeri. Inoltre la manutenzione dell'intero sistema può essere effettuata con delle modifiche localizzate sui server.

Tra gli svantaggi di questo tipo di soluzione dobbiamo notare che il server, essendo centralizzato, è necessario al funzionamento dell'intera rete. Di conseguenza il server rappresenta un "single point of failure". I sistemi Client/Server sono stati molto studiati dal mondo accademico e industriale, pertanto le soluzioni centralizzate peer-to-peer offrono pochi spunti di sviluppo teorico.

Anello

Un unico server centralizzato non riesce a gestire alti carichi di lavoro e una soluzione piuttosto comune richiede l'utilizzo di un cluster di macchine connesse ad anello, come illustrato in Figura 2.1(b), in grado di funzionare come un server distribuito. A differenza di altre topologie, quella ad anello richiede, nella maggior parte dei casi, che le macchine risiedano fisicamente nello stesso luogo. Eccezione fatta per una nuova proposta di topologia, denominata *Circle* [35], che propone una connessione ad anello distribuita, basata su una tabella hash decentralizzata, su piattaforma *JXTA* [31]. In [35] ogni nodo si occuperà di mantenere una parte dell'indice globale delle risorse e una serie di puntatori a nodi seguenti. Il prezzo da pagare in termini di risorse e di banda, per mantenere l'integrità della rete, essendo geometrica, risulta altissimo nel caso medio. Appare quindi una soluzione valida solo per piccole reti controllate.

Gerarchica

I sistemi gerarchici, illustrati in Figura 2.1(c), hanno una lunga storia in Internet, anche se, per le difficoltà che derivano da una struttura così rigida, vengono soppiantati da topologie di altro tipo. Esempi celebri di questa topologia sono il Domain Name Service e il protocollo NTP. Non esistono reti peer-to-peer che riescano a trarre vantaggio da questa topologia.

Decentralizzata

I sistemi decentralizzati prevedono una sola tipologia di nodo che, svolgendo mansioni sia da server che da client, viene tipicamente denominato servent. Le reti decentralizzate, per loro stessa natura, non hanno dei "single point of failure", non essendoci alcun nodo privilegiato necessario al funzionamento. La rete è rappresentabile come grafo non diretto nel quale tutti i nodi hanno un numero di archi variabile. Nella Figura 2.1(d) si nota l'assenza del server che aveva caratterizzato la Figura 2.1(a), mentre risalta la presenza di nodi dello stesso tipo e grado.

Reti di questo tipo devono saper risolvere diversi problemi, il primo fra questi è appunto il problema della prima connessione: in sistemi decentralizzati puri non dovrebbe esistere nemmeno un punto fisso di entrata, purezza raramente rispettata nelle implementazioni. Un altro problema fondamentale riguarda il routing dei messaggi, che richiede la specifica

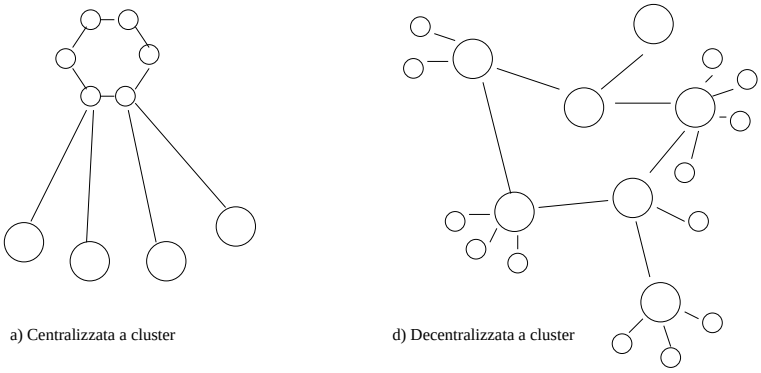


Figura 2.2: Topologie Ibride

di protocolli che offrano diverse garanzie. In particolare si richiede la raggiungibilità di ogni nodo da parte di un numero consistente e sufficiente di altri nodi, trovando un compromesso tra l'overhead sulle transizioni e la larghezza dell'orizzonte di visibilità. Per garantire la scalabilità, le reti decentralizzate devono operare degli accorgimenti speciali per ridurre la quantità di risorse (banda) utilizzate per mantenere la struttura, altrimenti soggetta a collasso.

In ambienti puri è importante anche riuscire ad escogitare dei meccanismi e delle strategie per permettere ai servent di valutare la reputazione di altri servent o direttamente delle risorse, in modo da riuscire a ponderare, con strumenti il più possibile solidi, il caso o meno di effettuare transazioni a rischio. Alcuni di questi problemi sono tuttora aperti e in questa tesi si mostrerà un protocollo che si propone di offrire una soluzione al problema della reputazione in ambito pseudoanonimo.

2.1.2 Topologie ibride

Spesso le topologie pure presentano degli inconvenienti che possono essere ridotti o addirittura eliminati operando delle modifiche alle strutture. Alcuni esempi di tipologie ibride sono quella *centralizzata a cluster* e *decentralizzata a cluster*.

- **Topologia centralizzata a cluster**

  Spesso le soluzioni centralizzate mostrano la loro insufficienza quando il carico richiesto alla struttura cresce oltre la soglia di gestibilità. Per ottenere una scalabilità migliore una delle soluzioni più valide è quella di utilizzare un cluster di server centralizzati e sincronizzati tra loro. In Figura 2.2(a) si mostra il caso di un cluster ad anello che funge da server centrale. Particolare cura deve essere posta nella sincronizzazione degli indici gestiti dai nodi del cluster, pertanto soluzioni di questo tipo spesso prevedono che i computer del cluster risiedano fisicamente vicini, in modo da garantire una banda comunicante larga e protetta. Per il resto sono assimilabili ai sistemi centralizzati.

- **Topologia decentralizzata a cluster**

  Uno dei modi per ottenere delle configurazioni ibride in ambiente decentralizzato è quella di permettere ad alcuni nodi di comportarsi, anche solo temporaneamente, con delle funzionalità di server: in quel particolare stato prendono il nome di *Supernodi*. In effetti, alcuni studi [62, 63] tendono a dimostrare che ambienti decentralizzati sprecano gran parte della banda utile in funzioni che potrebbero essere centralizzate in server di cache.

  I protocolli che usano questi meccanismi sono generalmente più complessi ma tendono ad avere dei risultati migliori di architetture decentralizzate pure, infatti alcune reti tradizionalmente pure [50] tendono ad integrare nei protocolli di base delle funzioni ibride per tentare almeno di arginare alcuni dei problemi sopracitati. Naturalmente non essendo facile stabilire quali servent possono eleggersi a supernodi né limitare il loro potere in modo da non intaccare l'anonimato dei client, ci sono ampi margini di sviluppo in questa direzione.

  Nella Figura 2.2(b) si rappresenta il tipo di rete proposta da Limewire [40] per risolvere alcuni dei problema citati in reti Gnutella (cfr 2.7): si nota la presenza dei supernodi, chiamati *UltraPeer*, che permettono la costituzione di connessioni particolari. Un nodo semplice connesso tramite questo protocollo ad un supernodo non necessita di altre connessioni strutturali, alleggerendo l'intera rete e permettendo una migliore scalabilità.

È da notare che, indipendentemente dalla particolare topologia adottata, in una rete peer-to-peer in generale è possibile identificare due tipi di connessioni: una strutturale e una diretta. Le connessioni strutturali servono a definire gli archi del grafo che rappresenta la rete e servono ad incanalare il protocollo di routing: attraverso queste connessioni vengono trasferite le informazioni relative alle risorse condivise da ogni nodo. La connessione diretta collega due nodi per il tempo necessario al trasferimento delle risorse, e mentre la connessione strutturale tende ad essere stabile e persistente, la connessione diretta avviene soltanto per il tempo necessario alla transazione.

Ci sono diversi esempi significativi, illustrati nel seguito di questo capitolo, che occorre studiare per poter capire quali siano le opportunità messe a disposizione dalle reti peer-to-peer. Vengono illustrati IRC (esempio storico di rete peer-to-peer), Napster (noto per le sue vicende giudiziarie), FreeNet (rete peer-to-peer anonima), KaZaA (implementazione commerciale molto efficiente) ed infine Gnutella (ambiente peer-to-peer puro). Di Gnutella vengono inoltre presentate le estensioni più importanti.

2.2 IRC

Internet Relay Chat [37] è uno dei protocolli più usati in rete ed è forse la rete peer-to-peer più antica, tanto da esistere da prima che fosse stato diffuso il concetto: nato per il chatting, permette di interagire nei forum con altri utenti. La connessione avviene mediante client a dei server, divisi in "reti". Alcune di queste reti storiche, come IRCNet, sono molto rigide nella gestione, altre invece lasciano diverso spazio alle libere iniziative. Capita così che si vengano a formare dei canali tematici, a volte contenenti fino a qualche migliaio di utenti, dove avviene il libero scambio di materiale. Molti nodi offrono delle risorse, e in alcuni casi esistono dei modi per ottenere degli indici generali, senza però coinvolgere i server. Il download avviene in modo diretto (per questo ha senso parlare di peer-to-peer), tramite protocollo DCC [59].

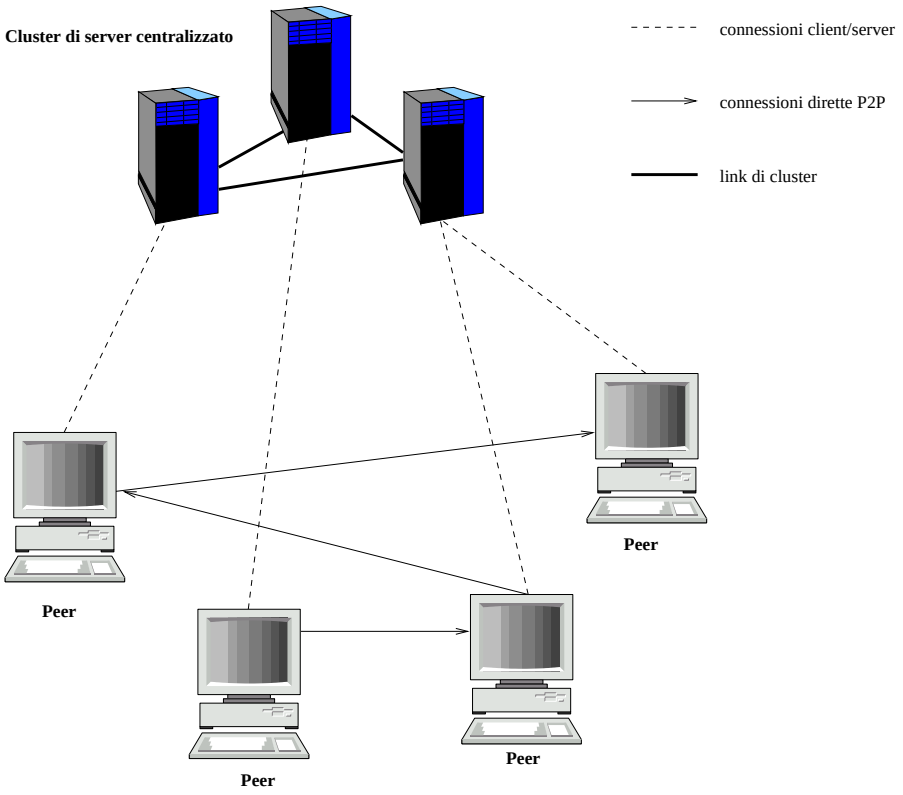


Figura 2.3: Architettura di Napster

2.3 Napster

Napster [50] è un prodotto che ha rivoluzionato il concetto di distribuzione di file musicali. Nato alla fine del 1999, in pochi mesi ha creato la più grande comunità virtuale di navigatori appassionati di musica, tanto da essere riconosciuto come vero e proprio fenomeno di massa.

Napster era in grado di facilitare lo scambio di file musicali MP3 [49] tra gli utenti di tutto il mondo. Perché il sistema funzionasse, tutti gli utenti dovevano scaricare un client che, collegandosi ai server di Napster, permetteva di autenticarsi, di condividere le proprie risorse musicali e di mettersi in contatto con gli altri utenti. Condividendo la parte del proprio disco rigido contenente i file musicali, si creava un canale di scambio in rete che permetteva di accedere a tutti i file disponibili in quel momento. Il protocollo di trasmissione, così come la tecnologia sviluppata per permettere lo scambio e la ricerca di file, è di proprietà di Napster. Elemento non trascurabile è che tutto avveniva in modo gratuito.

L'architettura di Napster ha una struttura ibrida centralizzata, come illustrato in Figura 2.3. La struttura prevede la presenza di un cluster di server accessibile da un unico punto di entrata. I computer del cluster sono connessi tra loro in modo da offrire ad ogni utente la visibilità completa delle risorse della rete. Per ogni utente venivano indicizzate le seguenti informazioni: login e password, utili per l'autenticazione della connessione, indirizzo IP e porta di provenienza ed elenco dei file MP3 condivisi. Ognuno di questi file veniva descritto dal percorso di accesso sul disco rigido dell'utente, dal nome, dalla dimensione e dalla qualità.[1] C'erano due modi per effettuare le ricerche. Il primo tipo di interrogazione prevedeva parole chiave, limitazioni sulle dimensioni, specifiche di qualità e forniva come risultato l'elenco delle risorse soddisfacenti le richieste. Per ognuno di questi risultati veniva elencato: login dell'utente, indirizzo ip e porta,

[1] Per qualità si intende la frequenza di campionamento e il bit-rate, rilevabili in modo automatico da parte del software.

velocità di connessione e i dettagli del file condiviso. Effettuata la scelta avveniva la connessione diretta che coinvolgeva i due peer, che a quel punto scambiavano la risorsa in modo autonomo. L'altro tipo di interrogazione, invece, prevedeva la possibilità di scorrere l'elenco dei file messi a disposizione di ogni singolo utente, conoscendone lo pseudonimo.

A questi sistemi era stata aggiunta la possibilità di comunicare attraverso una chat con altri utenti, sia dentro a dei forum, sia in modo diretto. Questo ha permesso il formarsi di comunità virtuali tematiche. Una strategia che veniva spesso praticata dagli utenti era quella di esplorare le risorse personali di un utente di cui era nota la validità dei gusti musicali, in modo da riuscire ad espandere la propria conoscenza. In sostanza, ognuno portava come propria reputazione l'elenco delle risorse condivise.

È ovvio che il numero di iscritti e i milioni di file scambiati nei mesi di attivazione di Napster hanno sconvolto il mercato discografico e la reazione delle case discografiche, già alla ricerca di nuovi modelli di business con i vari portali musicali, non si fece attendere: cominciarono una serie di azioni legali per fermare il perpetuarsi della situazione. La storia si è chiusa, almeno per il momento, con la partnership tra il colosso discografico Bertelsmann e Napster. Il risultato è che, l'accesso a Napster non è più gratuito, ma si deve pagare un canone mensile per aver diritto all'abbonamento È comunque importante osservare che per le case discografiche non è stato un danno completo, poichè il mercato delle vendite è, in quei tempi, cresciuto: una spiegazione di questo fenomeno è data dal fatto che uno delle conseguenze del fenomeno Napster è che milioni di persone nel mondo hanno avuto l'opportunità di espandere le loro conoscenze musicali in modo molto semplice ed economico, facendo nascere in molti la passione per la musica.

La ragione per cui Napster ha potuto essere disattivato è legato fortemente al tipo di architettura usata: il protocollo prevedeva che appena connessi al cluster di server, tutti di proprietà di Napster, venisse trasferita la lista completa delle risorse offerte dal client stesso. Napster, in altre parole, era stato ritenuto complice del traffico illegale, proprio per esserne stato a tacita conoscenza.

Nel periodo di apogeo di Napster nacquero dei cloni, uno dei quali era OpenNapster [52, 79]. Questo progetto Open Source si prometteva di offrire gli stessi risultati di Napster ma non riscosse mai lo stesso successo dell'originale, finendo in fretta soppiantato da nuove tecnologie più promettenti.

Ultimamente un successore di Napster che riscuote un discreto successo è WinMX [77] che sfrutta una topologia ibrida decentralizzata a cluster per offrire risorse multimediali.

2.4 FreeNET

Il progetto FreeNET [16] nasce nel 1999 per il desiderio di garantire il diritto inalienabile della libertà di stampa e di comunicazione in Internet. Permette a chiunque di pubblicare e leggere informazioni di ogni genere nell'anonimato più completo, grazie all'utilizzo di sistemi crittografici forti. Nessuno controlla FreeNET, per definizione, nemmeno i suoi creatori: ciò significa che il sistema è protetto contro ogni tipo di manipolazione o di controllo da parte di governi o di autorità superiori. La filosofia alla base di questo progetto è piuttosto diffusa in rete e assume che il diritto più importante da preservare nella nostra società sia il diritto di comunicare e condividere le proprie idee senza alcun tipo di censura e controllo, essendo, la conoscenza, alla base di un qualunque sistema politico democratico. Questo diritto in effetti è messo seriamente in discussione

da diversi provvedimenti che, in tutti i paesi del mondo, cercano di arginare l'anarchia intrinseca di Internet ponendo, laddove possibile, dei controlli e delle censure. Questo problema è esteso anche ad altri campi, primo fra tutti la fonia: si pensi ad esempio a Echelon [24], un sistema per l'ascolto sistematico di chiamate telefoniche allo scopo di controllare comunicazioni pericolose per l'integrità dell'economia statunitense. Naturalmente, consci del fatto che esistono software che permettono molto facilmente di leggere quasi tutta la e-mail che viaggia nella rete[2], si capisce come sia sentito, specialmente in paesi totalitari, la necessità di avere uno spazio assolutamente incontrollabile.

La rete, ovviamente decentralizzata, fornisce una piattaforma di pubblicazione navigabile tramite web in grado di ospitare, come un WWW parallelo, un insieme anche molto grande di pagine. Il sistema si appoggia su un client Java da scaricare ed installare sul proprio computer che associa, ad una porta TCP locale, il servizio di navigazione e permette, per default, la navigazione in FreeNET solo dalla macchina stessa sulla quale gira il programma.

La prima versione funzionante e pubblica di questo software è stata rilasciata sotto licenza GPL [33], che garantisce il diritto a chiunque di avere e modificare i sorgenti del programma. Tutti i processi interni che garantiscono il funzionamento di FreeNET sono completamente anonimizzati e decentralizzati, cosa che rende estremamente difficile per un attaccante distruggere, manomettere o censurare del materiale o di prendere controllo dell'intero sistema. FreeNET garantisce la privacy degli utenti rendendo virtualmente impossibile scoprire quali informazioni viaggiano sulla rete e impossibile risalire alle pagine lette da un particolare utente. La sicurezza è garantita da un substrato crittografico sufficiente a proteggere le transazioni e la manomissione dei pacchetti che compongono i flussi di informazione necessari al funzionamento della rete stessa.

L'efficienza purtroppo è piuttosto scarsa. Il meccanismo è forse troppo complesso e risente del fatto di essere sviluppato in Java e di essere ancora in fase di test. Esistono tuttavia dei metodi di replicazione dei dati per cui ogni informazione viene spezzata, crittata e replicata per la rete, in modo da ridurre i problemi derivanti dall'uscita improvvisa di rete di un nodo. FreeNET richiede, in teoria, per recuperare un dato da una rete di dimensione n un tempo paragonabile a $log(n)$, ma in realtà, attualmente la rete è praticamente non utilizzabile, data la sua lentezza.

In FreeNET ogni file è identificato da una chiave binaria ottenuta applicando una funzione di digest (attualmente SHA-1 [9, 56]) al suo contenuto, alla sua descrizione o ad una chiave crittografica associata ad una risorsa. Per recuperare un file è necessario conoscerne la chiave di ricerca, in molti casi non indovinabile: pertanto, file di cui si è persa la chiave diventano definitivamente irreperibili.[3] Segue la descrizione dei tre tipi di chiavi binarie, differenti per scopi e per complessità.

- *keyword-signed key (KSK)* : la più semplice, crea la chiave sulla base di una descrizione del file data dall'autore. Per permettere il recupero del file basta conoscere la sua descrizione, semplice da ricordare e comunicare. Presenta tuttavia un inconveniente: essendoci un unico namespace è possibile che due file differenti abbiano la stessa descrizione, e quindi lo stesso nome, rendendoli entrambi irreperibili.

- *signed-subspace key (SSK)* : permette la creazione di namespace personali, delle directory distribuite nelle quali poter costruire sottodirectory e aggiungere file. Questo spazio è identificato da una coppia di chiavi pubblica/privata create nell'atto della sua costituzione. Ogni

[2] Il protocollo SMTP [61, 55], usato nella stragrande maggioranza dei casi è privo di qualunque tipo di codifica crittografica. Esiste un'estensione crittografica chiamata TLS [36] che tuttavia non è ancora molto diffusa

[3] Il che significa che è possibile nascondere in rete dei file reperibili soltanto da chi ne conosca la chiave d'accesso.

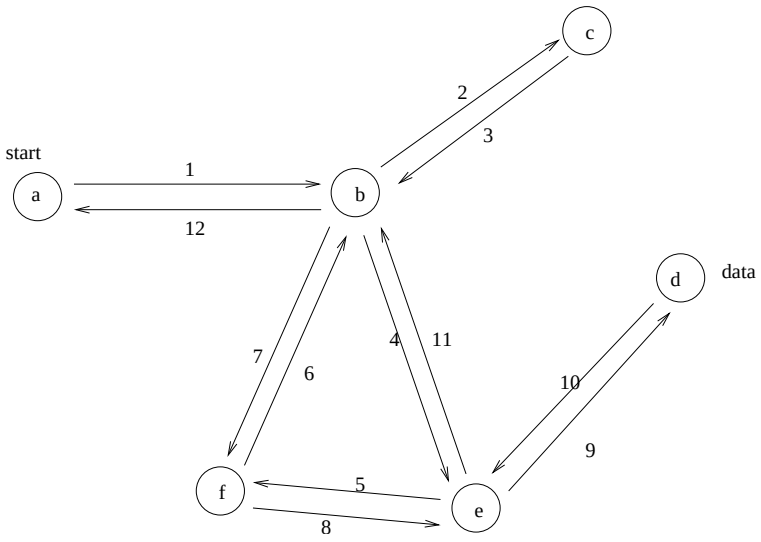


Figura 2.4: Recupero dei file in FreeNET

nuovo file creato in questo spazio assume come identificativo l'unione mediante funzione XOR della chiave di ricerca dello spazio e di una descrizione del file, che quindi può simulare sottodirectory (Es: text/philosophy/sun-tzu/art-of-war, text/philosophy/confucius/analects).

- *content-hash key (CHK)* : la chiave viene calcolata mediante il digest del contenuto del file, rendendo virtualmente unico il suo descrittore. Questo metodo viene utilizzato per immettere file modificabili e per permettere di spezzare il file in molte parti, utile essenzialmente per evitare le limitazioni di banda e di carico.

Per recuperare un file in FreeNET occorre quindi calcolarne la chiave, in un caso ricostruibile mediante la sua descrizione (keyword-signed key), negli altri (signed-subspace e content-hash key) deve essere data. Ottenuta la chiave, che chiameremo k, questa viene cercata nell'indice delle chiavi del proprio nodo a. Qualora questa non sia presente, si seleziona nell'indice la chiave successiva k_1, di cui si conosce il nodo b che possiede il file ad essa associato. Questo nodo viene interrogato, come illustrato in Figura 2.4, e se possiede il file relativo alla chiave k lo restituisce. Altrimenti l'interrogazione viene propagata verso gli altri nodi, usando lo stesso meccanismo, fino ad aver compiuto un certo numero di tentativi configurabile.[4] I nodi che ricevono dei file li conservano ad uso futuro. Tendenzialmente, questo comportamento, tende a raggruppare file delle stesse directory negli stessi nodi e a replicare i file più letti. Per l'inserimento di nuovi file viene utilizzata una tecnica simile: come prima cosa viene propagato un messaggio che cerca la presenza di una chiave uguale a quella prodotta dal nuovo file, conflitto che causerebbe danni potenziali. Successivamente, ottenuto un messaggio "all clear" da parte dei nodi contattati, il file viene propagato e inserito nei nodi interpellati nella richiesta di conflitto. Per inserire un nuovo nodo nella rete, questi deve generare un nuovo identificativo di nodo e mandare un annuncio pubblico. A questo punto questo nodo viene inserito nelle tabelle di routing dei vari nodi contattati cominciando così a farsi carico di registrare i nuovi file immessi, come specificato nella procedura relativa.

Esistono molti altri modi per ottenere l'anonimato e la distribuzione del materiale in reti non sottoponibili ad alcuna censura, come dimostrano protocolli come *Tangler* [76], *Free Heaven* [21] e *Publius* [44].

[4] Tipicamente il valore è fissato a 25

2.5 FastTrack: KaZaA, Morpheus e Grokster

Una piccola rivoluzione fu portata da KaZaA [39] che, in collaborazione con FastTrack, produsse un sistema basato su architettura ibrida a supernodi estremamente efficiente e funzionale.

A differenza di Napster FastTrack non mantiene un indice centralizzato e non attua alcun tipo di filtro sui contenuti. Come Napster è un sistema chiuso, richiede una registrazione iniziale (gratuita) e per la connessione bisogna fornire login e password.

KaZaA, Morpheus e Grockster sono client a distribuzione gratuita che utilizzano la stessa tecnologia, acquistata in licenza da FastTrack, azienda olandese impegnata da anni nello sviluppo di soluzioni peer-to-peer. Tutte e tre le implementazioni condividono la stessa popolazione di utenti e si offrono mutua visibilità.[5] A differenza di Napster, che offriva supporto solo per la condivisione di file musicali MP3, queste implementazioni lasciano distribuire qualunque tipo di file: audio, video, immagini, documenti e programmi. Ad ognuno di questi file sono associati dei metadati specifici, così da riuscire a riconoscere attributi come titolo, artista, categoria, lingua, risoluzione, ed altri. Inoltre sono presenti due caratteristiche molto interessanti che risultano estremamente utili in reti vaste ed eterogenee: *SmartStream* e *FastStream*. *SmartStream* offre un rimedio ad un tipico problema di Napster: la difficoltà a completare il download dei file prelevati in rete. Spesso accadeva infatti che una risorsa smettesse di essere disponibile e il file rimanesse scaricato solo in parte. Questa caratteristica di FastTrack permette invece di ritrovare automaticamente la stessa risorsa da qualche altro nodo e riprendere il download dal punto in cui si era fermato. *FastStream*, d'altra parte, permette di scaricare contemporaneamente lo stesso file da sorgenti differenti, così da massimizzare la velocità di download.

Questi sistemi adottano diverse tattiche per massimizzare il numero di server presenti in rete e la quantità di risorse globali. In particolare:

- I file scaricati diventano subito disponibili alla comunità, così che ogni nodo diventa punto di ridistribuzione di ogni file scaricato.
- I programmi si installano automaticamente in memoria e se chiusi si minimizzano e continuano a funzionare in background.
- Il sistema riparte automaticamente ad ogni riavvio
- Il sistema mette a disposizione un lettore MP3. La ragione di questo è che buona parte degli attuali utilizzatori di Napster usi soltanto il player MP3 integrato.

Negli ultimi tempi KaZaA è diventato tristemente famoso per aver cominciato a distribuire insieme al client anche una serie di programmi (che si installano automaticamente) che tendono a raccogliere il maggior numero di informazioni possibili sull'utente, per ricerche di mercato. Per questa ragione, i suddetti programmi vengono denominati *Spyware*.

Lo stack peer-to-peer di FastTrack, scritto in C++, è la base dell'architettura che si autodefinisce di prossima generazione, distribuita e auto organizzante. Questa architettura, contrariamente al concetto di peer-to-peer, richiede la presenza di un server centrale responsabile di mantenere la registrazione degli utenti, autenticarli nel sistema e fornire degli indirizzi di nodi esistenti ai client che si connettono alla rete. Il riferimento dei nodi necessario per la connessione è soltanto

[5] Recentemente KaZaA ha cambiato protocollo, impedendo agli utenti di Morpheus di condividere con loro, come prima avveniva, le risorse

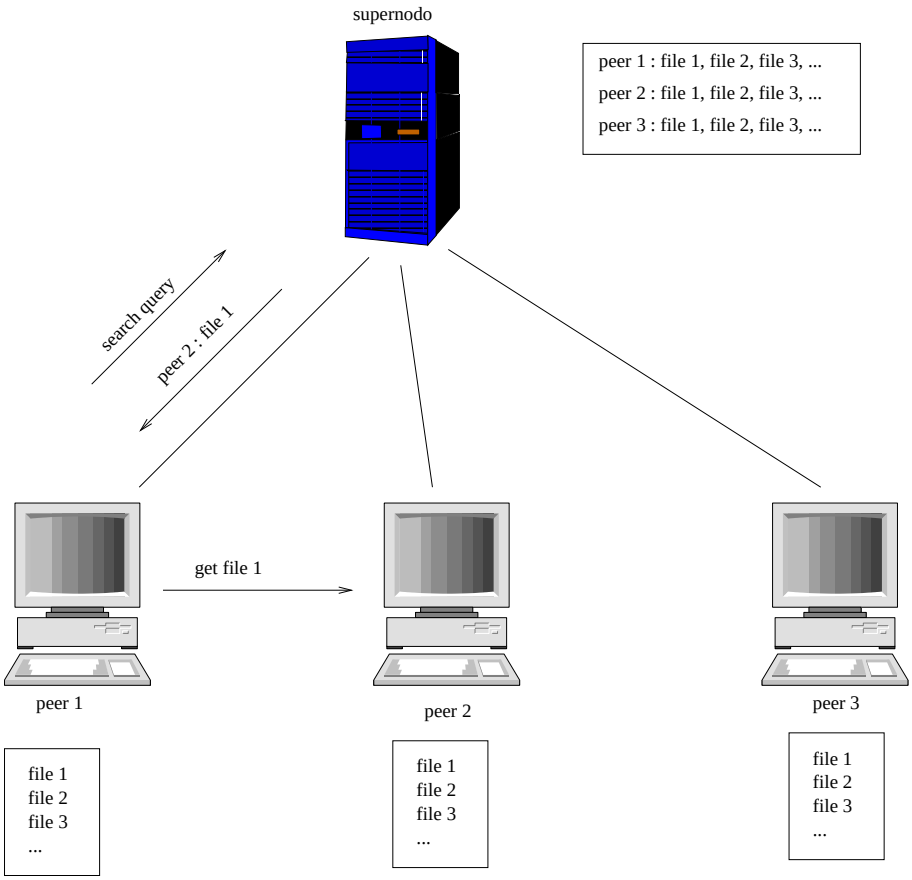


Figura 2.5: Schema di funzionamento di un Supernodo FastTrack

l'indirizzo IP, poiché la porta è sempre la 1214. I Supernodi sono la vera caratteristica innovativa dell'architettura perché organizzano, come dei concentratori, la propagazione e la ricerca degli indici. Ogni Supernodo infatti, raccoglie gli indici offerti dei vari nodi semplici che si connettono loro e si comportano come dei proxy, raccogliendo e inoltrando le richieste dei vari servent. Questo schema permette di ottenere delle velocità sorprendenti nella ricerca e fornisce la visibilità di un orizzonte molto vasto, paragonabile (e forse addirittura superiore) a quello offerto da strutture centralizzate.

I Supernodi vengono automaticamente eletti al loro stato privilegiato se viene rilevata una quantità considerevole di banda e di memoria, perché la gestione degli indici di un numero consistente di connessioni è oneroso. Usando questo meccanismo, che toglie a Morpheus e KaZaA le responsabilità legali della gestione degli indici, sono state registrate reti composte da centinaia di migliaia di nodi, numero addirittura superiore a quanto mai ottenuto da Napster.

Di fatto, quello che ogni nodo compie è di gestire in modo trasparente un Web server minimale sul proprio PC, essendo il protocollo HTTP quello usato nelle transazioni.

Purtroppo il protocollo KaZaA, pur essendo stato studiato e analizzato da Clip2, azienda che attualmente ha terminato l'attività, rimane protetto da copyright e inaccessibile a gran parte del mondo accademico.

2.6 Altri client P2P

Nel corso dei mesi sono nate decine di implementazioni diverse di reti peer-to-peer, molte di queste riscontrano soltanto dei brevi momenti di notorietà. Un breve elenco delle architetture più significative include le seguenti:

- *AudioGalaxy* [8] : dedicato allo scambio di file audio, si accede mediante pagina web. L'utente decide cosa scaricare da un indice centralizzato che applica dei filtri per evitare il diffondersi di materiale sotto copyright. Successivamente, un client preinstallato, chiamato satellite, riceve le istruzioni dal browser ed effettua la connessione peer-to-peer verso i nodi che posseggono i file trovati.
- *EDonkey2000* [25] : offre come caratteristica innovativa la possibilità di condividere del software non ancora completo, in modo da produrre delle catene di download in grado di propagare le risorse tra i vari nodi che le richiedono, almeno in teoria, in tempo quasi costante.
- *Direct Connect* [22] : richiama i concetti di IRC, offre un'interfaccia a centinaia di server, ognuno dei quali gestito autonomamente. Permette di effettuare delle ricerche generali.

2.7 Gnutella

Il 14 marzo 2000, alle 11:31 ora di New York, fu postato un messaggio su Slashdot [67] (un sito di news piuttosto popolare nell'underground hacker) che dichiarava che la divisione Nullsoft di AOL aveva appena rilasciato una versione OpenSource di un nuovo software denominato *Gnutella*, che si proponeva come alternativa libera a Napster. Tramite questo strumento si poteva accedere ad una rete nella quale era possibile condividere documenti e programmi, effettuare

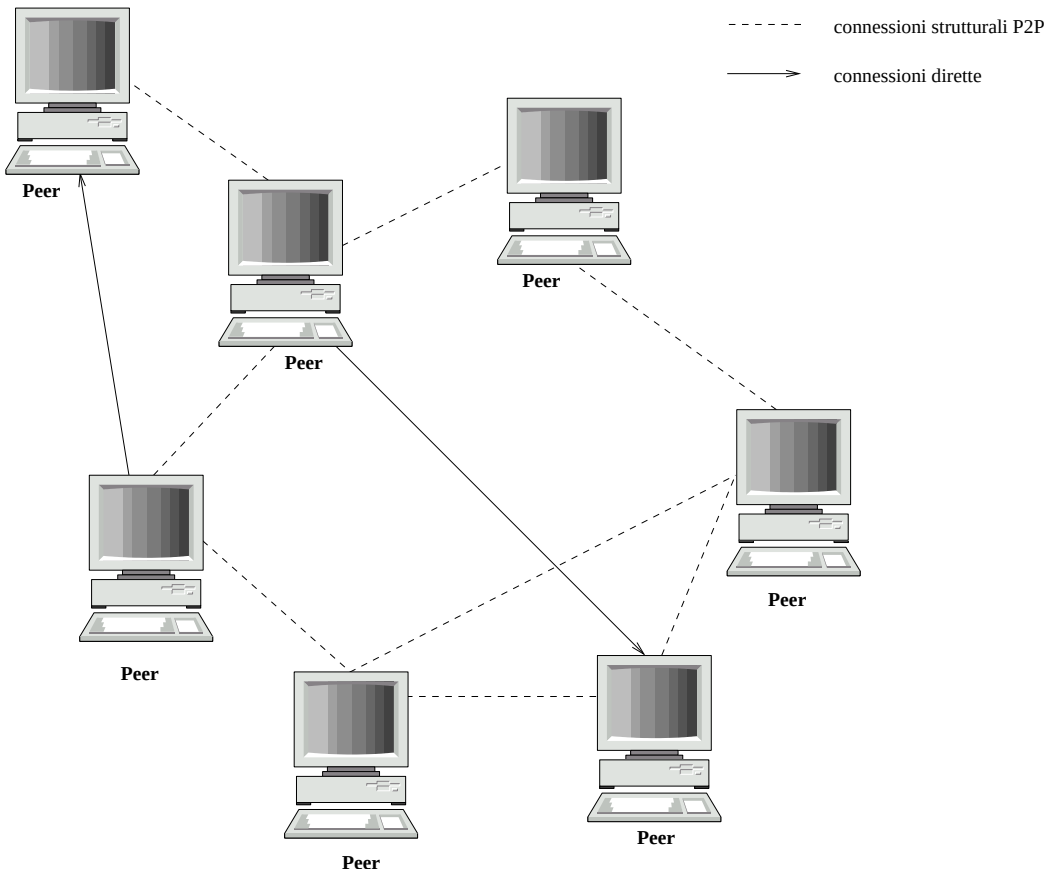


Figura 2.6: Architettura di Gnutella

delle ricerche e scaricare sul proprio computer i file così trovati. Le ricerche, a differenza di Napster, non erano limitate soltanto al tipo di file musicale MP3, bensì era possibile scambiare ogni tipo di file. Meno di diciotto ore dopo il rilascio di Gnutella, Wired News [78] (il bollettino della nota rivista del cyberspazio Wired) annunciava che Nullsoft, riconoscendo di aver prodotto un software ben più potente e potenzialmente pericoloso di Napster, già noto per aver indispettito non poche case discografiche, avrebbe ritirato immediatamente dal mercato il client. In effetti fu proprio a causa di alcune trattative con la EMI e la Warner Music che AOL fu costretta ad una così repentina inversione di tendenza. Ma ormai il software era stato immesso negli ingranaggi insondabili della rete e per quanto fu disponibile soltanto poche ore, qualche migliaio di persone avevano già scaricato il programma dal sito. Alcuni di loro, affascinati dal progetto, ne produssero dei cloni, garantendo sempre la compatibilità con il protocollo stabilito dal client originale. Fu necessario per questo un vero e proprio lavoro di reverse engineering che portò successivamente alla formalizzazione del documento Gnutella Protocol [73], attualmente considerata la base di un qualunque sviluppo dell'argomento. Così, con la comunità Internet che cominciava ad usare con regolarità questi cloni e con il fatto che circolavano per la rete copie non autorizzate del software originale, di fatto nasceva e cominciava a crescere una rete di applicazioni compatibili Gnutella che si distanziava dal modello Napster per la sua natura decentralizzata. In questi due anni questa rete è cresciuta in modo significativo ed ha finito per consolidarsi in una rete stabile.

Gnutella è uno dei pochissimi modelli peer-to-peer completamente decentralizzato. Ogni servent è connesso ad altri servent in un numero tipicamente compreso tra 1 e 15. La struttura della rete che si viene a creare è quindi un grafo sparso di cui non è garantita la connessione. Non essendoci nessun server privilegiato non esiste un accesso principale. Infatti qualora un servent desideri

connettersi alla rete gli è sufficiente trovare l'indirizzo IP e la porta di uno qualunque dei nodi della rete Gnutella. Una volta connesso ad un nodo può essere ottenuta una lista dei nodi attivi attualmente sulla rete e pertanto si può aumentare il numero di connessioni. Tante connessioni aiutano a migliorare la visibilità della rete a scapito di un consumo più alto di banda. Il problema della prima connessione è stato risolto pubblicando su un sito noto una lista aggiornata in tempo reale dei nodi attivi in quel momento (node caching). Una soluzione di questo tipo tuttavia comporta degli inconvenienti poiché certe volte, essendo limitato il numero di connessioni che ogni client tende a voler mantenere, occorre del tempo per trovare un nodo disponibile all'aggancio. Questa ricerca, data una lista, avviene per tentativi, e spesso richiede anche una decina di minuti. Un altro problema legato a questa soluzione sta nel fatto che la mortalità dei servent può essere piuttosto alta, pertanto una lista del genere rischia di invecchiare molto rapidamente. Una soluzione migliore è stata di creare un certo numero di nodi fissi, dei veri e propri portali di accesso alla rete. Questi nodi pubblici, il cui indirizzo in taluni casi è persino cablato nel codice dei servent, è disponibile nei siti più rappresentativi sull'argomento [29, 30].

Un servent Gnutella si connette quindi alla rete stabilendo una connessione con un altro server facente parte attualmente della rete tramite connessione TCP/IP. La porta standard in uso dalla maggior parte delle implementazioni è la 6345, anche se non è codificato in nessun protocollo. Creata questa connessione, il richiedente manda un messaggio ASCII di questo tipo:

```
GNUTELLA CONNECT/<versione del protocollo> \n\n
```

La versione standard del protocollo è 0.4 ma esistono diverse implementazioni che supportano un protocollo più avanzato (0.6, discusso in seguito), che permette di specificare eventuali funzionalità aggiuntive dei client. Tale funzionalità permette di ottenere un meccanismo di handshaking in grado di garantire la miglior connessione possibile e l'utilizzo di tutte le possibili estensioni non definite negli attuali protocolli.

Il server che desiderasse accettare la connessione deve rispondere con la seguente stringa, sempre codificata in ASCII:

```
GNUTELLA OK\n\n
```

Tipicamente si tende a limitare il numero di connessioni in entrata e in uscita, in modo da garantire una connettività sufficiente ma non eccessiva alla rete. Oltre a questa ragione ne esistono diverse altre per le quali può essere rifiutata una connessione. Ad esempio un client potrebbe non essere compatibile con un particolare protocollo e non accettarne la connessione.

Una volta connessi la comunicazione e lo scambio di informazioni avviene tramite messaggi che possono essere di tre tipi:

- **Multicast**

  Questi messaggi devono essere propagati attraverso l'infrastruttura di rete e vengono letti e interpretati da tutti i servent. Tali messaggi servono sia per effettuare delle ricerche (messaggi `Query`), sia per segnalare la propria presenza nella rete cercando di ottenere nel contempo una stima della sua estensione (messaggi `Ping`). Ogni nodo, come già detto, è connesso ad un certo numero di altri nodi e, in presenza di questo genere di messaggi, ha compito di propagarli verso tutti i nodi (ad eccezione del nodo da cui è arrivato il messaggio).

- **Unicast**

  I messaggi unicast utilizzano la rete Gnutella e servono per mettere in comunicazione due nodi. Tipicamente questi messaggi sono risposte a messaggi multicast, come ad esempio messaggi `QueryHit` (in risposta a `Query`) e messaggi `Pong` (in risposta a `Ping`). Un altro tipo di messaggio unicast è `Push`, che serve a mettere in comunicazione diretta nodi protetti da Firewall.

- **Diretto**

  Messaggi diretti tra nodi avvengono senza l'utilizzo dell'infrastruttura Gnutella e servono a mettere in comunicazione diretta due nodi che tipicamente devono scambiarsi dei file. Questi messaggi servono essenzialmente per gestire e per trasferire risorse.

2.7.1 Descrizione dei messaggi

I messaggi multicast generati da un nodo vengono inoltrati lungo tutte le connessioni attive: i nodi che li ricevono li rispediscono verso tutte le loro connessioni (ad esclusione di quella da cui il messaggio è giunto). Ogni messaggio è dotato di un contatore di Hop, ovvero di passaggi tra nodo e nodo e di un campo (TTL) che definisce dopo quanti Hop il messaggio deve essere eliminato. Il campo TTL, infatti, viene decrementato ad ogni passaggio ed al raggiungimento del valore zero impone al nodo che riceve il messaggio la cancellazione del messaggio stesso.

Per il fatto che spesso un pacchetto, per la natura non centralizzata della rete, tende a passare più volte dallo stesso nodo, risulta necessaria l'introduzione di un identificativo di messaggio atto al riconoscimento dei pacchetti già inoltrati.

Tutti i messaggi permessi da Gnutella sono riassunti nella figura 2.7: ogni messaggio scambiato in rete deve essere preceduto da un Description Header. Sotto ogni campo di dimensione fissata, viene riportata la sua misura della sua lunghezza espressa in bytes. Di fianco ad ogni nome di messaggio c'è la codifica esadecimale del suo Payload Descriptor. Tutti i valori dei campi sono codificati in *little endian* [26].

I messaggi Multicast e Unicast, quelli che in sostanza usano l'infrastruttura della rete, hanno tutti quanti il medesimo *Description Header*, che si antepone al messaggio specifico e che garantisce le specifiche del livello 2 (Network) nel modello ISO/OSI. Infatti il Description Header è composto come segue:

- DescriptorID (16 bytes)

  Il DescriptorID è l'identificatore di messaggio e viene generato casualmente. Ogni nodo tiene traccia di tutti i DescriptorID che incontra, in modo da riconoscere messaggi di un determinato tipo già inoltrati e per restituire le risposte verso l'origine che ha generato la domanda. Esiste infatti la non remota possibilità che alcuni messaggi, muovendosi attraverso un grafo ciclico non diretto, arrivino allo stesso nodo per due archi differenti. Quindi è compito del nodo stesso riconoscerli e lasciar passare solo il primo.

- Payload Descriptor (1 byte)

  Questo campo serve a descrivere il tipo di messaggio che segue. Attualmente sono utilizzati i seguenti valori: 0x00 (`Ping`), 0x01 (`Pong`), 0x40 (`Push`), 0x80 (`Query`) ,0x81(`QueryHit`).

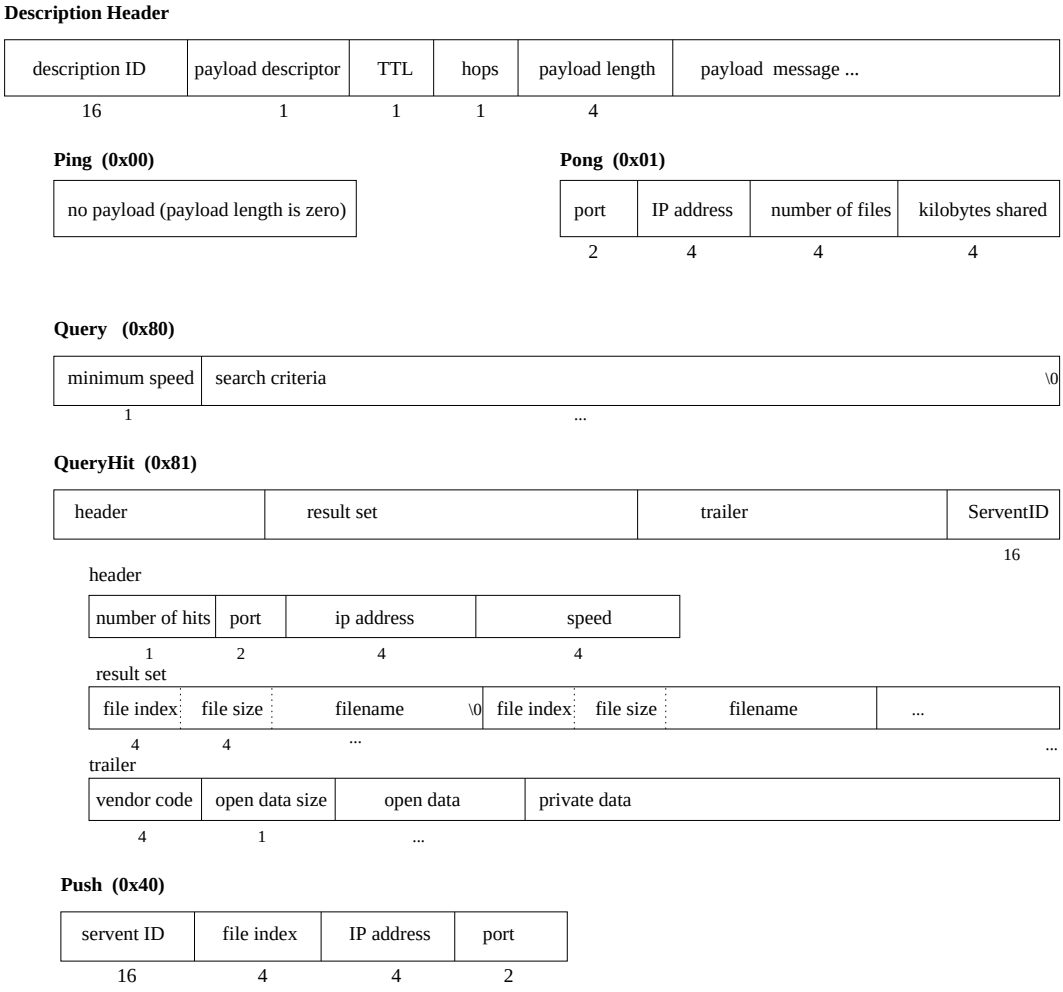


Figura 2.7: Messaggi Gnutella.

- TTL (1 byte)

  Time To Live. Ogni messaggio ha una vita prefissata e determinata dal numero di Hop effettuati: alla creazione di ogni pacchetto viene fissato questo campo ad un valore tipicamente intorno a 7 o 8. Ogni servent decrementa questo valore prima di inoltrare il messaggio e non lo inoltra affatto quando il valore raggiunge 0. Dal momento che il TTL è l'unico meccanismo esistente per eliminare pacchetti dalla rete, risulta estremamente importante, per le implementazioni, evitare che questi valori vengano abusati. Infatti valori troppo grandi tendono ad appesantire l'intera rete [63] e diverse implementazioni, per disincentivare gli abusi, eliminano messaggi il cui valore del campo TTL supera un certo valore.

- Hops (1 byte)

  Ad ogni passaggio da nodo in nodo, questo valore viene incrementato. Si noti che per ogni messaggio la somma dei valori TTL e Hops è uguale al valore che aveva assunto il campo TTL alla creazion del messaggio stesso. Questa condizione può essere definita in modo formale come segue:

  $TTL(0) = TTL(i) + Hops(i)$

  dove $TTL(i)$ e $Hops(i)$ sono rispettivamente i valori dei campi TTL e Hops all'$i$esimo Hop, con $i > 0$.

- Payload Length

  In questi quattro byte viene rappresentata la lunghezza del messaggio nel pacchetto. La dimensione massima di ogni messaggio, pur essendo in teoria 2^{32} byte (4 Gb), nella maggior parte delle implementazioni prese in considerazione viene fissato a 64 Kb.

Ping

I messaggi `Ping` non hanno associato alcun carico e hanno lunghezza 0. Un ping è quindi rappresentato soltanto da un Descriptor Header il cui campo Payload Descriptor e il cui campo Payload Length siano entrambi fissati al valore 0.

Un servent usa i Ping per conoscere lo stato della rete: un servent che riceva un pacchetto di questo tipo è tenuto a rispondere con un pacchetto Pong, che ne contiene l'indirizzo e l'ammontare delle risorse condivise da quel particolare nodo.

Non c'è alcuna specifica ufficiale che descriva le modalità e i tempi d'utilizzo di questo tipo di messaggi, ma ogni nodo dovrebbe tendere a minimizzarli all'indispensabile. Diversi studi [62, 63] hanno verificato che la maggior parte dei pacchetti che transitano su una rete sono di questo tipo.

Sono attualmente allo studio diverse soluzioni per arginare il problema della presenza percentuale eccessiva dei pacchetti `Ping`, a cominciare dalla soluzione proposta da LimeWire [34, 40].

Pong

Il `Pong` è un messaggio di risposta al `Ping` ed è caratterizzato dai seguenti campi:

- Porta (2 byte)

  Questo valore rappresenta la porta alla quale il servent risponde alle domande di connessione in entrata.

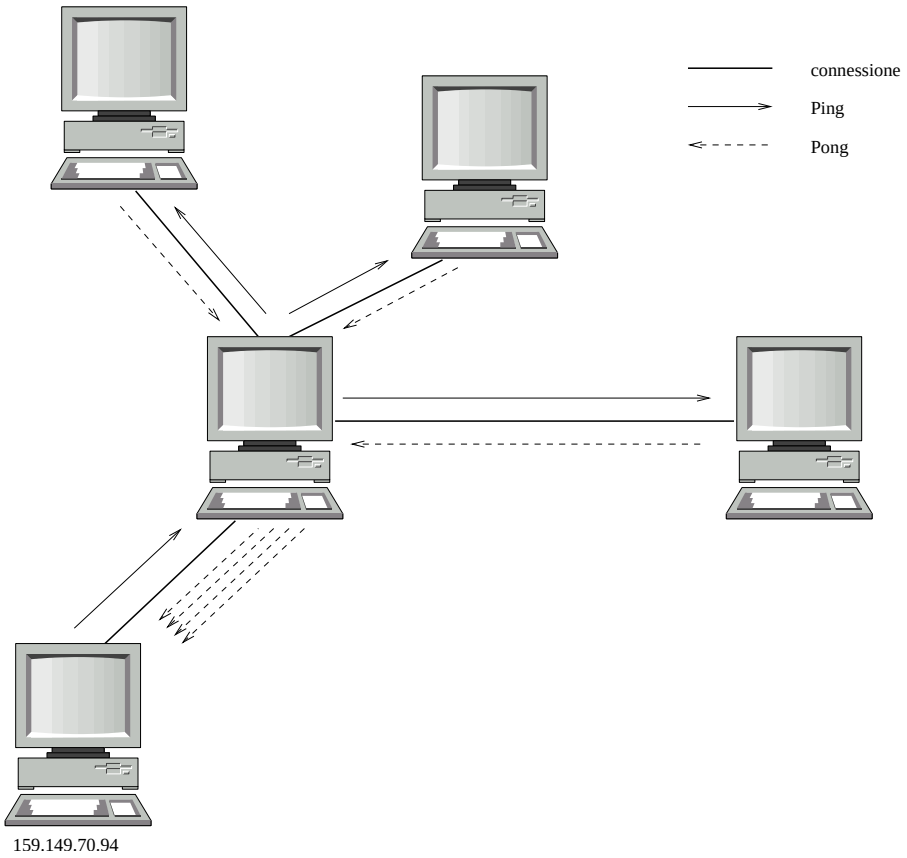


Figura 2.8: Routing dei pacchetti Ping e Pong

- Indirizzo IP (4 byte)

  L'indirizzo IP viene rappresentato in formato big endian [26].

- Numero di file condivisi (4 byte)

  Numero dei file condivisi dall'host identificato dall'IP e dalla porta specificati.

- Numero di KB condivisi (4 byte)

  Dimensione totale dei file condivisi espressi in Kb.

In Figura 2.8 si vede come ogni nodo raggiunto da un messaggio `Ping` risponda con un diverso `Pong`, generando un alto traffico di rete. [6]

Query

I messaggi `Query` servono ad effettuare delle ricerche nella rete. Sono messaggi Multicast ed ogni nodo che li riceve è tenuto a confrontare i criteri di ricerca con il proprio insieme di risorse condivise per verificare la presenza di file che risponderebbero alle esigenze descritte. Questi messaggi sono caratterizzati dai seguenti campi:

- Velocità minima (2 byte)

[6] Certi servent, il cui indirizzo è pubblico, possono comportarsi da Cache di indirizzi, in modo da fornire a chiunque si colleghi, un insieme di indirizzi di nodi della rete al momento validi. In questi casi è ammessa la spedizione da parte dell'host di un alto numero di messaggi di `Pong` in risposta ad un solo `Ping`.

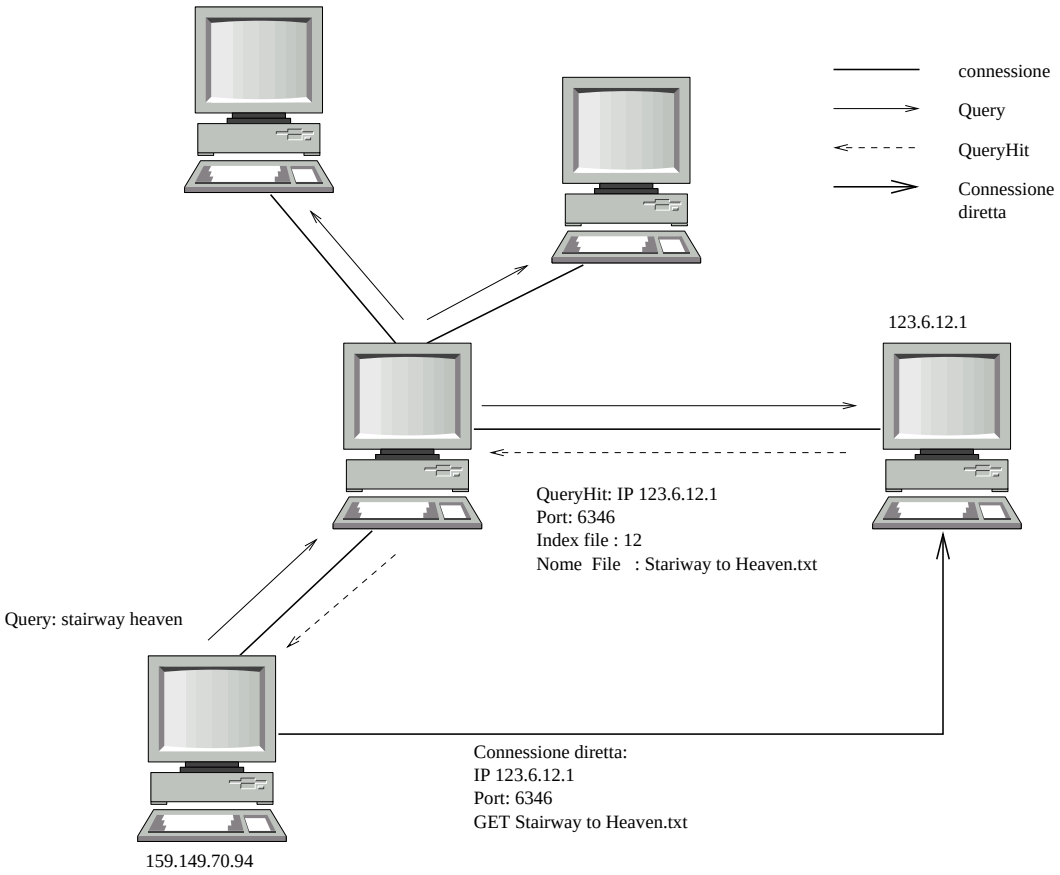


Figura 2.9: Routing di una sessione Query/QueryHit con download

La velocità minima, espressa in Kb/s, indica il limite inferiore di velocità richiesta ai servent che rispondono.

- Criteri di ricerca (null terminated string)

  Questo campo, di tipo stringa, delimitato dal carattere 0x00, è composto dall'insieme delle parole che devono essere presenti nel file di risposta.

QueryHit

In risposta ai messaggi `Query` ci sono dei messaggi unicast `QueryHit` che contengono tutte le specifiche necessarie per riuscire ad effettuare la transazione da parte di un client.

In Figura 2.9 è illustrata una sessione di ricerca nella quale vengono utilizzati messaggi `Query` e `QueryHit`.

- Numero di risultati (1 byte)

  Numero di record nel campo Risultati.

- Porta (2 byte)

  Porta del servent che risponde.

- Indirizzo IP (4 byte)

  Indirizzo IP del server in risposta.

- Velocità (2 byte)

  Velocità espressa in Kb/s.

- Risultati (...)

  Questo campo contiene un certo numero di record (fissato dal campo *Numero di risultati*). Ogni record è definito dalla seguente struttura:

  - Indice del file (4 byte)

    Valore assegnato dall'host rispondente che viene usato per identificare univocamente una risorsa all'interno del database delle risorse condivise.

  - Dimensione del file (4 byte)

    Dimensione espressa in Kb.

  - Nome di file (double null terminated string) Il nome del file che risponde alle richieste formulate nel campo *Criteri di ricerca* del messaggio `Query` relativo e che corrisponde all'indice specificato nel campo *Indice del file*. Questo campo è terminato con due caratteri 0x00.[7]

- Identificatore di Servent (16 byte) Ogni servent sulla rete deve essere identificato da una sequenza di 16 byte derivanti, in modo non meglio specificato dall'indirizzo IP. Molte tra le più importanti implementazioni utilizzano invece un numero casuale, fisso per ogni sessione ma non persistente, chiamato GUID. Questo identificatore (che nel seguito verrà chiamato `ServentID`) oltre ad essere essenziale per la comunicazione di nodi protetti da firewall, avrà un ruolo importante nella proposta di estensione di protocollo `P2PRep`.

I messaggi `QueryHit` vengono generati soltanto in risposta a `Query` e contengono soltanto proposte strettamente inerenti al criterio di ricerca ricevuto. Il Descriptor ID nel Descriptor Header deve contenere lo stesso valore del rispettivo campo del messaggio `Query`, in modo da permettere al messaggio di ritornare a destinazione tramite il protocollo di routing.

Push

Ottenuto un messaggio di `QueryHit` il nodo può stabilire una connessione diretta per effettuare la transazione effettuando un collegamento TCP all'indirizzo e alla porta specificata. Qualora un servent non possa ricevere delle connessioni dirette può comunque condividere delle risorse utilizzando un meccanismo che permette di spedire il file.

Qualora si voglia scaricare delle risorse da un servent non in grado di ricevere connessioni dirette, si può mandare dei messaggi `Push` che esprimono la volontà di ottenere determinati file, identificati dagli indici associati. I messaggi `Push` sono così composti:

- Identificatore di Servent (16 byte) Questo campo specifica a quale servent è indirizzato il messaggio.

- Indice del file (4 byte)

  Questo indice è il codice utilizzato dal server per identificare una risorsa e compare nel nel file record del messaggio `QueryHit` relativo.

[7] Lo spazio compreso tra i due caratteri null può essere utilizzato per eventuali scopi privati [73]

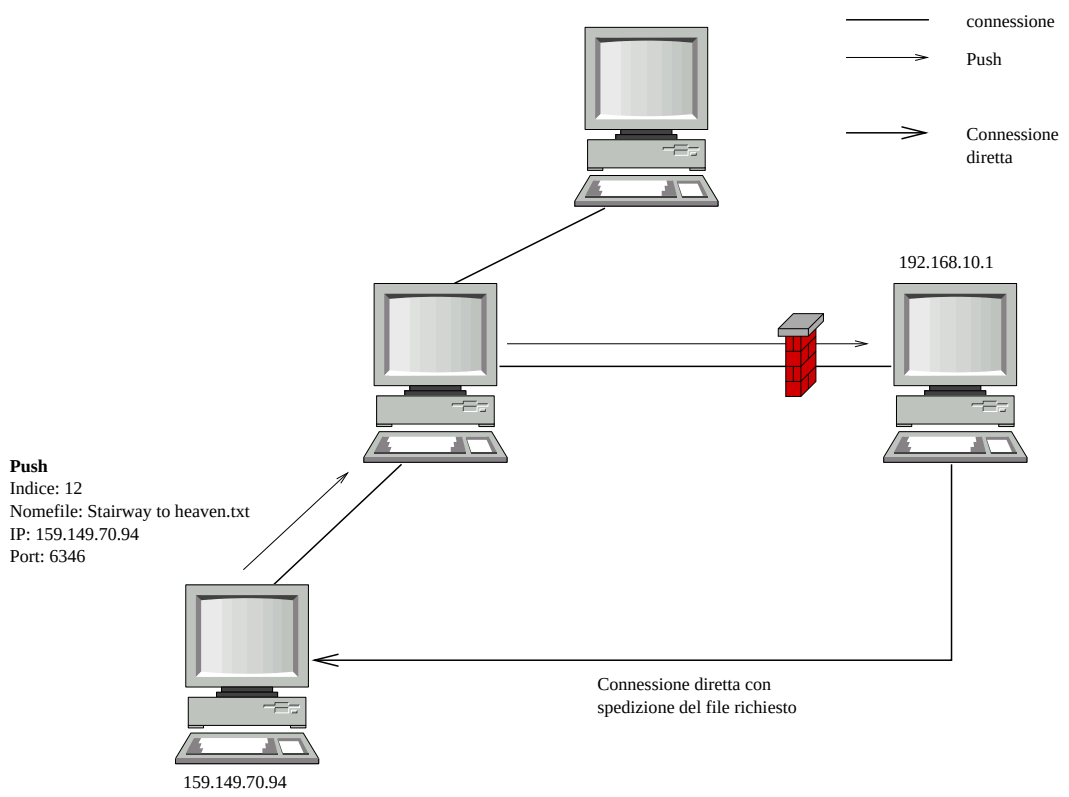


Figura 2.10: Routing dei pacchetti Push e download

- Indirizzo IP (4 byte)

  L'indirizzo del client a cui il server dovrà spedire (push) il file richiesto, codificato in big endian.

- Porta (2 byte)

  Porta di accesso al servizio.

Un servent che riceve un messaggio `Push` si preoccupa di spedire la risorsa soltanto nel caso in cui l'Identificatore di Servent corrisponda al proprio, negli altri casi si limita ad inoltrarlo come un comune messaggio Multicast.

Una sessione tipica di utilizzo del protocollo di `Push` è esemplificata nella Figura 2.10: qualora un servent (159.149.70.94) voglia scaricare una risorsa (Stairway to Heaven.txt) da un servent protetto da un firewall (192.168.10.1) deve chiederne la spedizione tramite un messaggio `Push`.

Messaggi `Push` permettono quindi di far comunicare servent quando uno solo dei due è protetto da un firewall ma diventa inutile qualora lo siano entrambi. Esiste una proposta di tunnellizzazione [57] che permetterebbe a due servent di comunicare servendosi dell'appoggio di un terzo servent, tuttavia non ancora implementata.

2.7.2 Routing

La natura peer-to-peer di Gnutella richiede che ogni nodo indirizzi i pacchetti che transitano in modo appropriato. I messaggi Multicast (`Ping`, `Query`) vengono inoltrati a tutte le connessioni attive eccetto a quella da cui il messaggio è arrivato, purchè siano verificate le seguenti condizioni:

Indice	*Dimesione*	*Nome del File*
12	5123	"Stairway to Heaven.txt"

Figura 2.11: Esempio di record

1. il messaggio deve essere integro, ovverosia i valori nei campi devono rispettare lo standard definito.
2. il valore TTL deve essere maggiore di 0
3. il messaggio non deve essere stato precedentemente inoltrato.

Per verificare l'ultima condizione, ogni implementazione tiene traccia degli identificativi dei messaggi(`DescriptorID`) che sono passati per ogni connessione. È importante che questa tabella contenga il tipo e l'origine del messaggio per il fatto che i messaggi di risposta a messaggi Multicast possano ritrovare la strada da cui sono giunti. I messaggi Unicast (`QueryHit`, `Push`, `Pong`) hanno infatti un indirizzo di destinazione uguale all'identificativo del messaggio al quale viene data risposta: un servent che voglia rispondere ad un messaggio di `Query` deve costruire un pacchetto di risposta che abbia lo stesso identificativo dell'originale. Per evitare che il pacchetto venga eliminato dalla rete è necessario che ogni nodo, oltre all'identificativo del messaggio, tenga traccia anche del tipo e della connessione di origine. Questo permette ai messaggi di risposta (unicast) di ritrovare la strada da cui sono state generate le rispettive ricerche.

2.7.3 Scaricamento dei file

Una volta che un servent abbia ricevuto risposta ad una ricerca, dopo aver selezionata la risorsa da scaricare, viene stabilita una connessione diretta al nodo relativo. Il protocollo usato per trasferire i file è il protocollo HTTP 1.0 [11].

```
GET /get/<indice del file>/<nome del file>/ HTTP/1.0
Connection: Keep-Alive\r\n
Range: bytes=0-\n\r
User-Agent: Gnutella\r\n
\r\n
```

Supponendo ad esempio di voler scaricare un file descritto dal record di Figura 2.11, si utilizzerebbe il seguente protocollo:

```
GET /get/12/Stairway to Heaven.txt/ HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\n\r
User-Agent: Gnutella\r\n
```

Il server, nel caso in cui effettivamente il file gli appartenga e abbia disponibilità ad effettuare nuove connessioni risponderebbe come segue:

```
HTTP 200 OK\r\n
Server: Gnutella\r\n
Content-type: application/binary\r\n
Content-length: 5123
```

2.7.4 Estensioni al protocollo

Una delle caratteristiche peculiari e straordinariamente interessanti del protocollo Gnutella è la sua semplicità; tutto il protocollo si articola con cinque messaggi e con un meccanismo di routing essenziale. Poterlo implementare in poche righe di codice e con poche risorse ha certamente contribuito alla sua espansione e alla sua distribuzione (naturalmente sono state proposte decine di estensioni [60] e di proposte, nate e sviluppate con scopi diversii).

Un'estensione [66] richiede di utilizzare uno o più byte del `DescriptorID` per gestire le versioni del protocollo in modo da riuscire ad identificare i nuovi client in grado di supportare particolari funzioni, ma tende a non essere compatibile con i vecchi client, fornendo quindi misurazioni in parte falsate. Un'altra [43] propone l'immissione nel protocollo di un nuovo messaggio (`Bye` 0x02) utile per chiudere le connessioni in modo elegante e per fornire la ragione dell'uscita dalla rete, utile soprattutto per fini sociali. L'estensione HUGE [48] si propone di associare ad ogni file un'identificazione assoluta (URN), indipendente dal nome e dalla locazione ma soltanto dal contenuto del file.

Altre estensioni importanti, che verranno trattate nel seguito con maggior attenzione, comprendono: la possibilità di inserire nei messaggi `QueryHit` dei dati privati, un protocollo di handshaking avanzato, un protocollo per la definizione di nuove estensioni, uno schema Ping/Pong per ridurre il loro impatto sulle performance della rete, un modello per la gestione efficiente delle query mediante routing semantico, un modello per includere nelle ricerche metadati in XML ed infine un protocollo per l'inserimento dei Supernodi nella rete Gnutella.

Dati privati

Una forte spinta all'innovazione e alla personalizzazione dei messaggi viene data da case come Bearshare [10] e LimeWire [40] che producono commercialmente client Gnutella. In particolare, dalla versione 1.3 di BearShare, sono stati introdotti diversi nuovi campi nei pacchetti `QueryHit`, con l'intenzione di fornire a chi genera la risposta ad una ricerca, informazioni aggiuntive sia sulle risorse stesse, sia sullo stato dei server a disposizione. In effetti fino a questo momento l'unica informazione utile sullo stato dei servent veniva data dalla loro indicazione nel campo dedicato alla velocità di connessione, ma niente poteva essere dedotto per quanto riguardava la loro effettiva responsività sulla rete. Vengono quindi introdotti dei campi utili a specificare se il server abbia o meno degli slot di connessione aperti e disponibili e un campo per specificare il tipo di software usato come client, utile per riconoscere la possibilità di applicare varianti efficienti specifiche di alcune implementazioni. In particolare viene usato lo spazio *trailer*, che è composto dai seguenti campi: *Vendor code*, *Open Data Size*, *Open Data* e *Private Data*.

- *Vendor Code* : 4 byte dedicati a contenere un identificativo mnemonico dell'implementazione. Ad esempio, LIME rappresenta LimeWire, BEAR BearShare e così via.
- *Open Data Size* : definisce, in un byte, la lunghezza in byte del campo successivo. Tipicamente ha come valore 2.
- *Open Data* : Questo spazio codifica informazioni aggiuntive che possono rivelarsi molto utili per valutare la qualità di un server. In Figura 2.12 si vede lo schema dei due byte che compongono questo campo.

  Flag byte 1

OpenData Flag byte 1

OpenData Flag byte 2

Figura 2.12: Campi OpenData

  - `UploadSpeed` : 1 se il corrispettivo campo del byte flag2 è significativo.
  - `HaveUploaded` : 1 se il corrispettivo campo del byte flag2 è significativo.
  - `Busy` : 1 se il corrispettivo campo del byte flag2 è significativo.
  - `Push` : 1 se il server è protetto da un firewall o non ha ancora accettato delle connessioni entranti.

  Flag byte 2

  - `UploadSpeed` : 1 se il campo Speed del messaggio `QueryHit` rappresenta la velocità media (espressa in KB/s) degli ultimi 10 upload effettuati dal server.
  - `HaveUploaded` : 1 se il server ha già completato con successo almeno un upload.
  - `Busy` : 1 se il server non è in grado in quel momento di accettare ulteriori connessioni.
  - `Push` : 1 se il corrispettivo campo del byte flag2 è significativo.

- *Private Data* : Spazio utilizzabile dalle implementazioni per scopi privati. BearShare lo usa per includere delle informazioni crittografate. La sua dimensione viene calcolata conoscendo la dimensione totale del messaggio e quella di tutti i restanti campi.

Questa estensione, privilegiando i nodi che si dichiarano liberi, veloci e ben configurati, permette di ridurre in modo significativo il numero di tentativi necessari per trovare un server disponibile a rilasciare le risorse.

Extensible Handshaking Protocol

Noto come protocollo 0.6, il Extensible Handshaking Protocol rappresenta un metodo di connessione esteso che permette di specificare eventuali funzioni aggiuntive, così da ottenere la miglior connessione possibile [34]. Supponiamo ad esempio che un'implementazione abbia la possibilità di migliorare la ricerca dei file o la velocità di trasferimento utilizzando qualche funzione avanzata di routing intelligente. Tramite l'handshake iniziale il client potrebbe capire, all'atto della connessione, se i propri vicini siano in grado di supportarlo.

La connessione ad handshake esteso inizia con una stringa di questo genere:

```
GNUTELLA CONNECT/0.6\r\n
User-Agent: LimeWire\r\n
Query-Routing: 0.1\r\n
```

Si noti che dopo il CONNECT ci sono le funzioni richieste per la connessione. Qualora il nodo accetti risponderà con un messaggio HTTP:

```
GNUTELLA/0.6 200 OK\r\n
```

In caso rifiuti la connessione può specificare la ragione con messaggi del tipo:

```
GNUTELLA/0.6 401 Unauthorized\r\n
```

oppure:

```
GNUTELLA/0.6 503 Service Unavaliable\r\n
```

Pur non essendo backward compatible, il Extensible Handshaking Protocol è stato accettato dalla comunità e diffuso al punto che ormai sono pochi i client che non supportano questo protocollo. Esiste un'ulteriore estensione a questo protocollo che prevede un vero e proprio handshake dei requisiti, così da ottenere la miglior configurazione possibile.

La connessione mediante questo protocollo si articola in diversi punti:

1. Il client stabilisce una connessione con il server.
2. Il client manda la stringa GNUTELLA CONNECT/0.6<cr><lf>.
3. Il client manda gli header che specificano tutte le proprie funzioni avanzate.
4. il server risponde con la stringa GNUTELLA/0.6 200 OK<cr><lf>.
5. il server manda la lista di tutte le proprie funzioni avanzate, nello stesso formato del punto 3.
6. Il client manda la stringa GNUTELLA/0.6 200 OK<cr><lf>.
7. Il client restituisce eventuali messaggi necessari alla sincronizzazione della connessione.

Questo meccanismo di connessione viene largamente usato dalle implementazioni avanzate come *BearShare* e *LimeWire*, in modo da sfruttare le loro funzioni avanzate, come ad esempio le connessioni `UltraPeer` (Supernodi) di *LimeWire*.

Gnutella Generic Extension Protocol (GGEP)

Il protocollo Gnutella Generic Extension Protocol [74] permette di aggiungere estensioni al protocollo 0.4, ormai considerato insufficiente per molteplici aspetti. GGEP infatti si propone di fornire uno standard per definire estensioni del tipo del trailer di BearShare e garantisce:

- L'inclusione di testo o di dati binari. Ogni estensione ha la necessità di aggiungere nuovi campi nei vari messaggi Gnutella.
- Fornisce ai produttori di software spazi privati per aggiungere i propri dati. Molte implementazioni, infatti, hanno la necessità di includere nei vari messaggi dati privati utili alle loro estensioni.

- Estensioni multiple. Diverse estensioni devono infatti poter convivere nella stessa rete, dando ad ogni estensione degli spazi identificabili in modo univoco all'interno dei messaggi.
- Analisi lessicale non ambigua. Si vuole garantire che non ci siano molte interpretazioni possibili ai messaggi.

Purtroppo il protocollo non ha ancora avuto il riconoscimento che merita, infatti nessuna delle estensioni qui descritte lo utilizza.

Ping/Pong Scheme

Una delle accuse rivolte alla scalabilità di Gnutella dipende dalla necessità di gestire una quantità di messaggi `Ping` che cresce esponenzialmente con il numero di connessioni e che finisce per saturare gran parte della banda diponibile [63]. Lo schema tradizionale di gestione dei `Ping`, infatti, consuma quasi il 50% della banda di Gnutella. Inoltre gli indirizzi ottenuti dai messaggi `Pong` spesso non accettano nuove connessioni, rendendo difficile connettersi alla rete.

La soluzione [15] a tale problema proposta da LimeWire si basa sul fatto che i messaggi `Pong` servono ad espletare due funzioni:

1. Restituire un certo numero di indirizzi di macchine presenti in rete in modo da riuscire ad aggiungere connessioni.
2. Fornire un'indicazione dell'orizzonte di visibilità della rete.

Il nuovo schema proposto si pone l'obiettivo di ridurre in modo significatico la banda usata e di garantire risposte valide in grado di accettare effettivamente connessioni. L'unico difetto è che risulta più difficile fare una buona valutazione dell'orizzonte.

L'idea chiave dello schema è di mantenere in una memoria cache i messaggi `Pong`, per evitare il broadcasting troppo rapido di `Ping`. Inoltre viene limitato il numero di messaggi `Pong` inoltrati e i `Ping` vengono gestiti tramite multiplexer.

Lo schema Ping/Pong è un insieme di tre soluzioni, che prevedono la limitazione e la cache dei messaggi `Pong` e il multiplexing dei messaggi `Ping`.

Limitazione sui Pong Vengono inoltrati i messaggi `Pong` solo dei servent che non sono protetti da firewall, cioè di quelli che sono potenzialmente in grado di accettare le connessioni. Viene quindi selezionato e spedito un insieme eterogeneo rispetto al numero di Hop effettuati di una decina di messaggi `Pong`.

Cache dei Pong La limitazione sui `Pong` non basta a ridurre il problema nel caso in cui arrivi un numero alto di `Ping`. In questo caso, se il servent ricevente tenesse memoria degli ultimi `Pong` ricevuti potrebbe, senza inoltrare alcun messaggio `Ping`, restituire quanto richiesto come illustrato in Figura 2.13. Naturalmente, per garantire la qualità delle risposte la cache si deve svuotare ogni pochi secondi.

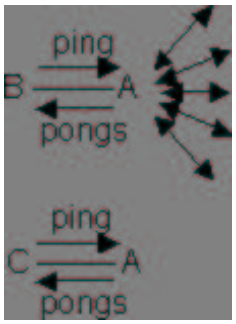


Figura 2.13: Cache dei messaggi Pong

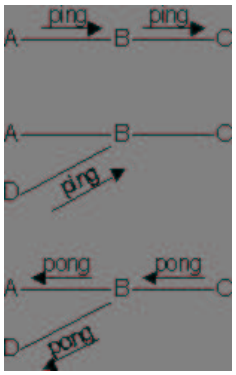


Figura 2.14: Multiplexing dei Ping

Multiplexing dei Ping Nel caso in cui un servent non abbia ancora riempito la propria cache e due nodi gli richiedano quasi contemporaneamente di inoltrare un `Ping`, il ricevente, come ilustrato in Figura 2.14, risponde ad entrambi con lo stesso insieme di messaggi `Pong`.

Quest'estensione risponde effettivamente alle esigenze più pressanti della rete e verrà a breve inclusa in una nuova versione di LimeWire.

QueryRouting

Questa estensione [7, 65, 46] consiste in uno schema per migliorare la propagazione delle richieste convogliandole verso i nodi che possono effettivamente rispondere loro. Si usano come strumento di indirizzamento tabelle che riflettono il contenuto del database delle risorse, senza tuttavia renderlo noto. L'idea di fondo è quella di utilizzare una funzione applicata alla stringa di ricerca come indice di routing. Mediante un'apposita funzione di hash viene quindi associato ad ogni risorsa, un digest definito da stringa di bit. Applicando la funzione OR ai digest di tutte le parole che compongono le stringhe di ricerca[8] si può ricavarne una che, per quanto non possa essere funzione univoca della ricerca, dia una ragionevole indicazione sul suo contenuto.

Ogni nodo deve quindi calcolare il digest composito di tutte le proprie risorse. Supponiamo che il nodo N abbia nel proprio archivio di condivisione due file: { *crypto.manifesto.txt*, *costituzione.italiana.txt* }. Per ogni parola che costituisce i nomi dei file viene calcolato il digest, che supponiamo essere il seguente: { 00101001,10000100,10001000,000010001,10010100,10001000 }. Il digest composito N_D del nodo viene calcolato mediante la funzione OR:

$N_D = 00101001\ OR\ 10000100\ OR\ 10001000\ OR\ 000010001\ OR\ 10010100\ OR\ 10001000 =$

[8] Ogni stringa di ricerca può essere composta da un insieme di parole, ognuna delle quali deve essere presente nei file inclusi nella risposta

$= 10111101$

Ora supponiamo che un nodo M effettui la ricerca { *costituzione* } (con digest $D_1 = 000010001$) e che conosca il digest composito dell'archivio N_D del nodo N. Siccome

$D_1 \; AND \; N_D \; == \; D_1$

il nodo M può sperare che effettivamente N possegga la risorsa cercata. D'altra parte, se invece avesse utilizzato, come stringa di ricerca { *gnutella.exe* }, il cui digest composito D_2 fosse { 01001011 }, dal momento che

$D_2 \; AND \; N_D \; \neq \; D_2$

avrebbe la certezza che il nodo N sia sprovvisto della risorsa cercata.

Il broadcast dei messaggi `Query` potrebbe essere evitato se si potesse sapere quali risorse siano perseguibili, in modo diretto o indiretto per una data connessione. Lo schema garantisce la non esistenza di falsi negativi, cioè una ricerca viene sempre inoltrata verso i nodi che effettivamente possono fornirle risposta, mentre non è in grado di evitare la presenza di falsi positivi, che tuttavia sprecano soltanto un po' di banda senza mai comportarsi in modo peggiore di come si comporterebbe un client tradizionale.

Ogni nodo deve tuttavia propagare il proprio digest composito di archivio e si vedrebbe così recapitare un certo numero di digest, ognuno dei quali proveniente da una particolare connessione con un determinato TTL.

Lo schema originale del protocollo [46] nasce in un contesto ad albero, dove non ci sono cicli. Perché sia portabile in ambiente decentralizzato è necessario associare alle stringhe anche il TTL minimo per raggiungerle; in questo modo, per ogni nodo, ad ogni connessione viene associato un numero di digest pari al massimo TTL ammesso. Ogni ricerca percorrerebbe quindi soltanto un numero esiguo di archi, garantendosi al contempo la raggiungibilità delle risorse.

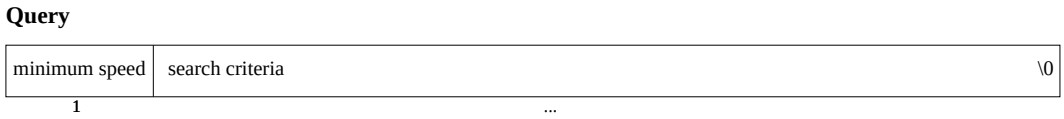
La funzione di hash proposta in [46] soddisfa i seguenti requisiti: produce una stringa di lunghezza costante, si implementa facilmente e permette di convertire digest di lunghezza arbitraria in equivalenti digest di altra lunghezza. Quest'ultima proposizione si formalizza come segue: data $h(k, m)$ una funzione hash che produce stringhe di lunghezza m a partire da una stringa k, conoscendo m ma non k, è possibile computare $h(k, m')$ con $m' \neq m$.

Per implementare questa estensione serve introdurre dei nuovi messaggi per la sincronizzazione dei digest compositi di archivio, che potrebbero sostituire, con particolari accorgimenti, i messaggi `Ping` e `Pong`.

Con una soluzione di questo tipo verrebbe ridotto in modo significativo il traffico dei messaggi `Query`, solo in parte recuperato dai nuovi messaggi necessari per lo scambio e la sincronizzazione delle tabelle di routing, mantenendo stretto riserbo sui file effettivamente condivisi.

Metadati in XML

Gnutella prevede che ogni file sia definito e ricercabile solo attraverso il suo nome e la sua dimensione, che spesso non sono sufficienti per descriverne il contenuto. È stata proposta un'estensione [71, 72] che permette di associare ad ogni file dei metadati che riflettono il contenuto e le informazioni aggiuntive utili, come ad esempio il bit-rate delle musiche e gli attori protagonisti dei film. Alcuni di questi metadati sono calcolabili direttamente dal file, altre richiedono l'intervento di un operatore umano.



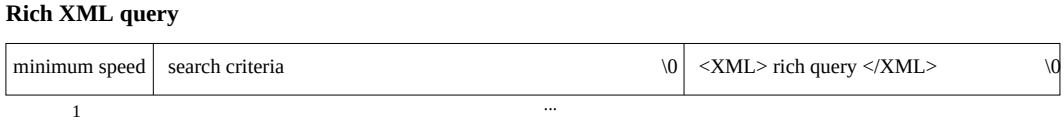


Figura 2.15: Schema del messaggio di query nell'estensione XML del protocollo Gnutella

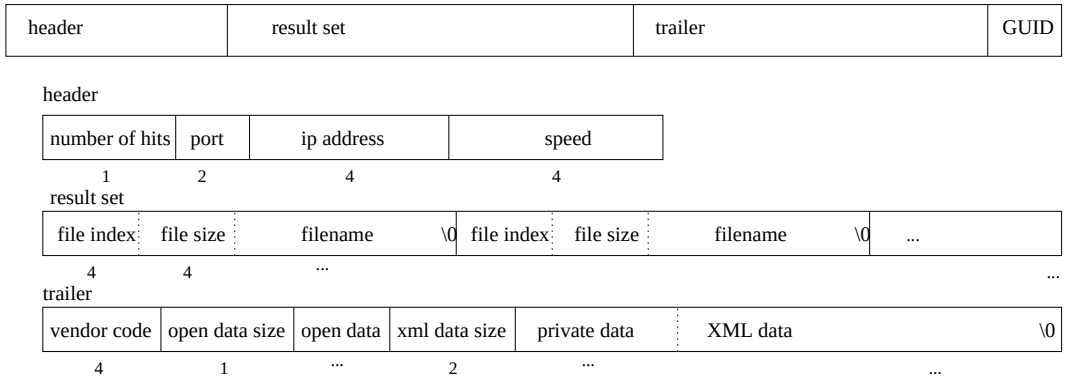


Figura 2.16: Schema del messaggio di risposta nell'estensione XML del protocollo Gnutella

Questi metadati vengono codificati con XML, in modo da riuscire ad associare ad ogni tipo di file uno schema XSD che ne definisca le possibili proprietà. Essendo questo schema definito mediante un URL, risulta molto semplice per un client estendere il servizio a tipi non previsti in una prima analisi, poiché basterebbe creare e rendere pubblico lo schema relativo. I nuovi messaggi, come illustrato nella Figura 2.15, prevedono uno spazio per inserire dei documenti XML relativi allo schema del tipo di ricerca che si effettua, nel quale vengono evidenziate le chiavi di ricerca nei vari attributi delle risorse.

Le risposte, il cui modello è rappresentato in Figura 2.16, saranno anche loro incluse in un documento XML allegato in un apposito spazio privato.

I campi dedicati al XML possono essere compressi con un consueto algoritmo `zlib` [19, 51], così da ridurre in modo considerevole lo spazio occupato.

Supponendo di voler cercare un libro di cui si conoscano il titolo, la casa editrice e l'autore, si potrebbe definire la ricerca in questo modo:

```
<books>
<book schema="http://www.limewire.com/schemas/book.xsd"
title="Big Bang "Origin of the universe"
publisher="ABC Publishing Co"
author="John Doe"
 />
</books>
```

Una risposta possibile potrebbe essere data in questa forma:

```
<"xml version="1.0"">
<books xsi:noNamespaceSchemaLocation=http://www.limewire.com/schemas/book.xsd>
<book  identifier="C:\shared\Big-Bang.txt "
title="Origin of the Universe"
author="John B Doe"
chapters="11"
genre="Non-fiction"
publisher"ABC Publishing Co"
price="$12.50"
comments="Extremely well written"
SHA1="c6b38bcdd3bb7755cce282f2fdd04512911e9ae0"
</book>
<book identifier="abc.txt"
                title = "Big Bang Universe"
                publisher="ABC Publishing Co LLC"
                author="John Doeky"
                chapters="14"
                comments="Not about the universe at all"
</book>
</books>
```

Le potenzialità dei metadati, implementati nelle ultime versioni di LimeWire [40], sono sfruttabili come strumento per gestire la reputazione dei nodi e delle risorse, come illustrato nel Capitolo 5.

UltraPeer

Le topologie ibride, rispetto alle architetture peer-to-peer pure, presentano un protocollo più complesso e difficile da mantenere ma anche un comportamento più efficiente. Gnutella, architettura pura, subisce alcuni limitazione alla scalabilità dovute proprio al fatto di non aver previsto alcun modo di avvalersi della profonda eterogenità dei nodi che la compongono e anzi subire dei rallentamenti globali a causa di nodi lenti. Ultrapeer [6] è un metodo per differenziare i nodi più veloci, dando loro l'opportunità di sfruttare al meglio le loro possibilità ed agire come concentratori di rete.

La Figura 2.17 mostra una tipica archiettura UltraPeer: si nota come alcuni nodi siano connessi alla rete tramite connessione UltraPeer tramite Supernodi.

Questa soluzione prevede che i nodi nella rete non siano omogenei, ma differenziati in tre tipi: i client tradizionali, le foglie (che necessitano di agganciarsi ad un supernodo), e i supernodi, che invece formano l'ossatura di una nuova struttura che serve ad irrobustire tutta la rete. Mentre le foglie hanno bisogno di mantenere soltanto una connessione con un supernodo, questi ultimi tenderanno ad avere fino ad un centinaio di foglie e una decina di connessioni con altri supernodi. I supernodi riescono a schermare quasi completamente le foglie, permettendo loro di sfruttare le potenzialità offerte dalla rete senza sprecare gran parte della banda per il suo sostentamento, completamente demandato al supernodo.

Per ottenere il mascheramento dei peer si utilizza lo schema Query Routing Protocol, col quale è possibile evitare di esportare i dettagli delle risorse condivise, garantendo riservatezza alle foglie.

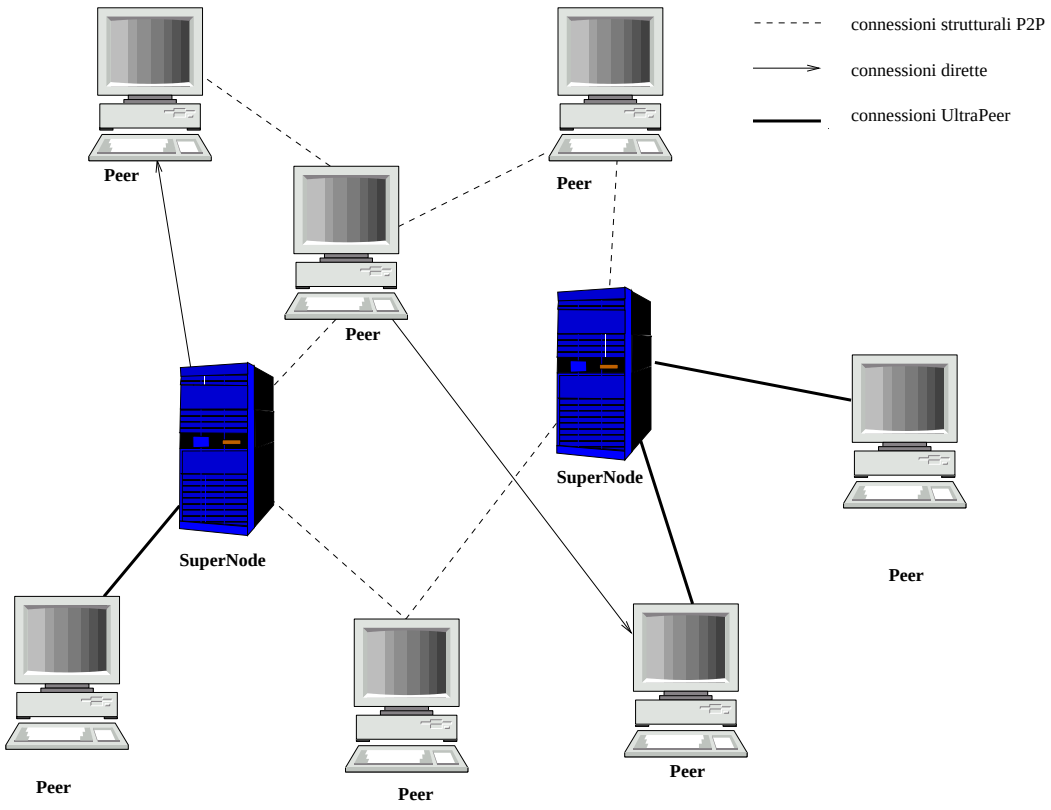


Figura 2.17: Architettura di una rete *Gnutella* con estensione Ultrapeer

La connessione tra Supernodi invece mantiene il vecchio protocollo, pur essendo possibile utilizzare QRP. In questo modo, i vecchi client Gnutella appaiono come se fossero dei Supernodi privi di foglie.

La connessione e il tipo di protocollo utilizzato vengono stabiliti durante la procedura di handshake del protocollo 0.6, durante la quale i nodi possono dichiarare di volersi comportare da foglie o da Ultrapeer.

Limewire implementa questa soluzione dalla versione 1.2. Questa soluzione è comparabile con quella adottata da FastTrack [28], della quale però non si conoscono le specifiche tecniche.

Capitolo 3

La reputazione negli ambienti P2P: un approccio al protocollo Gnutella

Chi dice la verità prima o poi viene scoperto

Oscar Wilde

3.1 Debolezze di Gnutella

Gnutella è un ambiente estremamente malleabile e dinamico, sufficientemente giovane ma abbastanza studiato per essere utilizzato come piattaforma di sviluppo per definire il protocollo di sicurezza che segue. Uno dei problemi che in questa tesi si vuole affrontare è dato dalla facilità con la quale, in questo tipo di ambienti, sia possibile la distribuzione dei virus e il dilagare di software illegale e pericoloso. Le ragioni principali di questo problema sono essenzialmente due e sono strettamente legate alla natura stessa del sistema: lo pseudoanonimato dei nodi e la varietà di risorse presenti in rete. I servent sono riconoscibili da un identificativo di sessione che, potendo cambiare ad ogni istanza e non essendo legato, se non in teoria, all'indirizzo IP, non fornisce sufficienti garanzie per essere considerato un riferimento assoluto dei servent. Nemmeno l'indirizzo IP può essere usato come identificativo univoco di servent. Infatti non solo capita che più servent abbiano lo stesso IP, ma anche che un utente abbia diversi servent posti su indirizzi diversi. Si consideri inoltre che rendere pubblico il proprio IP è sì necessario per effettuare delle connessioni, ma non lo è per la spedizione dei file, essendo possibile dichiararsi protetti da un firewall e prevedere che la spedizione effettiva del file avvenga da un indirizzo anonimo.[1]

Un malintenzionato che volesse propagare un virus in rete potrebbe facilmente dargli un nome di una risorsa comune, attendere che qualcuno la richieda, rispondere con un messaggio `QueryHit` dichiarandosi protetto da un firewall (ad esempio potrebbe dichiarare indirizzo IP **0.0.0.0** o **127.0.0.1**), attendere il messaggio `Push` e generare una connessione da un'altra origine per spedire il file infetto.

[1] Una tecnica per generare connessioni da indirizzi anonimi, complessa ma attuabile ancora su sistemi tipo Windown NT 4, si chiama Blind Spoofing ed è implementata in diversi strumenti disponibili in rete. L'attacco si basa sulla predicibilità dei sequence number usati nei pacchetti IP ed è necessario che il sistma operativo che gestisce lo stack TCP utilizzi numeri pseudocasuali prevedibili

Un'analisi del traffico di messaggi mostra che tra le risorse maggiormente richieste compaiono anche gli eseguibili, veicolo privilegiato per virus e Trojan horse.

In altri ambienti Internet, come ad esempio nel mondo WWW o negli archivi FTP, nel caso in cui venga riscontrata e verificata la presenza di materiale inadatto alla pubblicazione, perchè illegale od infetto, è possibile avvertire gli amministratori del sistema e, indagando negli archivi WHOIS, risalire ai nomi dei responsabili giuridici dei siti in questione. In Napster gli utenti scorretti potevano essere disabilitati. Al contrario non c'è alcun genere di controllo nè di verifica attuabile nelle reti Gnutella. Nemmeno le blacklist di indirizzi IP sono una valida contromisura. Infatti non c'è alcun modo di propagare queste informazioni in rete e, dal momento che molti servent si connettono da ISP con indirizzo dinamico, queste liste perderebbero molto rapidamente di valore.

In effetti è già apparso un worm in grado di sfruttare queste debolezze e propagarsi proprio grazie a questa rete: il worm `VBS.Gnutella`, spesso considerato impropriamente un virus, si diffonde copiandosi nelle directory utilizzate dai client Gnutella per la condivisione delle risorse. Il worm modifica quindi il file `gnutella.ini`, responsabile della configurazione del client, per permettere la condivisione dei file `.vbs`, in modo da potersi propagare.

Altri attacchi possono essere generati sfruttando la possibilità di rispondere facilmente a messaggi `Query` con `QueryHit` costruite appositamente per ottenere qualche effetto voluto: è possibile, per esempio, rispondere a tutti i messaggi `Query` con un messaggio `QueryHit` che soddisfi la domanda, ma che sia un riferimento ad un file che non rispecchi quanto dichiarato dal nome.

L'esigenza quindi di trovare un metodo per ridurre questi rischi è pressante nelle comunità peer-to-peer: in questa tesi viene presentata una proposta di protocollo, implementato specificatamente per Gnutella ma pensato per una generica rete peer-to-peer, che sfrutta come discriminazione dei nodi pericolosi la loro reputazione, raccolta mediante delle votazioni.

3.2 Protocollo P2PRep: aggiungere la reputazione al protocollo Gnutella

In Gnutella, ogni server ha associato un identificativo (`ServentID`) che viene comunicato agli altri durante l'invio di messaggi `QueryHit`, come stabilito dal protocollo. Il nostro approccio richiede la persistenza degli identificativi dal momento che è l'unico modo per mantenere la storia di un servent nell'arco di tutte le sue transazioni. La persistenza di un `ServentID` non influisce sull'anonimato dell'utente dal momento che si comporta come uno pseudonimo opaco e non è associato in modo inequivocabile a nomi o indirizzi IP. La persistenza di un `ServentID` non influisce sull'anonimato dell'utente dal momento che si comporta come uno pseudonimo opaco e non è associato in modo inequivocabile a nomi o indirizzi IP.

Come già è stato illustrato nel capitolo precedente, in un ambiente tipo Gnutella un servent p che desideri cercare una risorsa genera un messaggio broadcast (`Query`) e seleziona dalla lista di coloro che rispondono (che chiameremo *offerenti*) un nodo al quale si connetterà per prelevare i file scelti. La scelta tipicamente viene effettuata sulla base del nome della risorsa, della velocità dichiarata, ed eventualmente da dati aggiunti dall'implementazione (come ad esempio quelli definiti dall'estensione *BearShare* che permettono di selezionare nodi che abbiano effettivamente disponibilità di accettare nuove connessioni) e della dimensione. Si noti che non c'è alcuno strumento per valutare a priori la qualità di una risorsa. In certi casi, risulta ragionevole scegliere la risorsa più comune in rete, nell'ipotesi evoluzionista che venga più facilmente distribuita

la versione qualitativamente superiore. Inoltre, qualora la transazione non dovesse terminare in modo corretto, sarebbe sempre possibile proseguire lo scaricamento del file da un altro servent. Purtroppo questa tecnica palliativa non offre reali garanzie, e la quantità di file privi di senso con nomi allettanti distribuiti in rete ne è la dimostrazione.

Il nostro approccio, chiamato P2PRep, permette ad un servent p di chiedere alla comunità quale sia la reputazione di altri servent, tramite la spedizione di un messaggio broadcast (`Poll`) che esprime la volontà di indire una votazione. L'idea fondamentale è la seguente: ogni servent, una volta ottenuto un insieme di risposte ad una `Query` seleziona un sottoinsieme degli offerenti e genera un messaggio broadcast diretto a tutta la comunità, per ottenere informazioni sulla loro reputazione. Tutti i nodi riceventi possono rispondere esprimendo le loro opinioni, costruitesi nel tempo e con l'esperienza diretta, dando a p la possibilità di effettuare un miglior raffinamento della scelta iniziale e ridurre i rischi legati alle transazioni.

Verranno presentati due tipi diversi di protocollo, una versione *base* e una *avanzata*. La versione base assume che i votanti non dichiarino la loro identità. La versione avanzata richiede che ciascun voto sia accompagnato da un identificativo del servent votante, così da poter valutare la credibilità del votante. In effetti, con questo secondo meccanismo è possibile attribuire un peso differente ai voti, sulla base della credibilità stessa dei votanti. Naturalmente, essendo la qualità di un server non necessariamente correlata alla sua abilità di fornire delle indicazioni utili, si parlerà di due tipi di esperienze differenti: la reputazione vera e propria legata alla distribuzione delle risorse, e la credibilità, che riflette l'abilità di un nodo di esprimere giudizi utili e veritieri. Il protocollo viene leggermente complicato dal fatto che è necessario aggiungere delle condizioni per ridurre le possibilità di violazioni. Per ridurre la vulnerabilità ad attacchi, è prevista la possibilità di effettuare dei controlli sull'identità dei votanti tramite connessioni dirette.

Per riuscire ad ottenere la sicurezza sufficiente, è necessario utilizzare in modo massiccio la crittografia asimmetrica, così da riuscire a garantire l'integrità dei pacchetti (firma digitale) e la confidenzialità (crittazione). A questo scopo, il protocollo richiede infatti che l'identificativo di servent sia funzione della chiave pubblica di una coppia di chiavi persistente. In questo modo è possibile controllare l'identità di un servent verificando la sua conoscenza della chiave privata associata in modo indissolubile all'identificativo. La chiave pubblica, che può essere resa nota tramite messaggio `QueryHit`, può essere utilizzata infatti per crittare una sequenza casuale di dati, decrittabile soltanto dal possessore della rispettiva chiave privata. Ad ogni votazione, viene generata un'altra coppia di chiavi, in modo da massimizzare laddove possibile l'anonimato e rendere complesso risalire alle votazioni indette da un determinato servent.

Segue la descrizione del protocollo P2PRep, nelle sue versioni base e avanzato.

3.2.1 Protocollo di Base

Il protocollo base richiede che, come nel protocollo convenzionale di Gnutella, il servent alla ricerca di una risorsa mandi un messaggio `Query` indicando i dettagli della richiesta; i nodi che abbiano dei file in grado di soddisfarla rispondono con dei messaggi `QueryHit` contenenti, oltre al loro identificativo di servent, i riferimenti alle risorse cercate. In Gnutella, a questo punto, il servent p sceglie uno o più nodi dai quali prelevare le risorse. Il protocollo P2PRep, come illustrato in Figura 3.1(a), prevede invece un ulteriore passaggio utile al raffinamento della ricerca: il servent p seleziona da questa lista di servent S quelli che ritiene migliori, sulla base delle risposte ottenute e delle esperienze acquisite in precedenza. Questa scelta iniziale serve ad estrarre dall'insieme delle risposte un sottoinsieme di nodi. Tipicamente i nodi veloci e i nodi con i quali sono

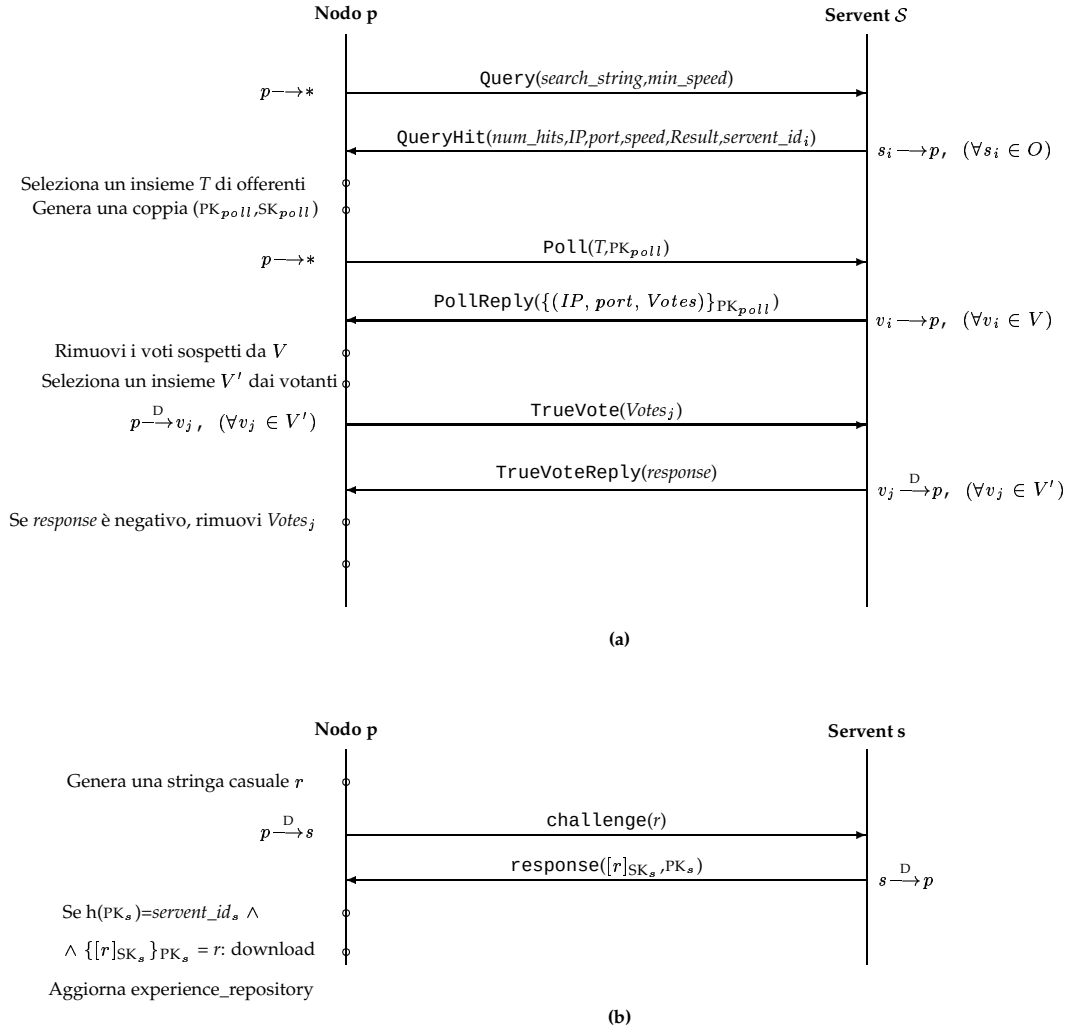


Figura 3.1: Sequenza di messaggi e di operazioni nel protocollo base (a) e interazione con il servent selezionato (b)

già state portate a termine transazioni soddisfacenti avranno maggior probabilità di essere eletti a candidati.

A questo punto il nodo *p* indice una votazione, chiedendo alla comunità di esprimere un giudizio sui nodi appartenenti all'insieme selezionato. Nel messaggio di votazione (`Poll`) vengono elencati gli identificativi di tutti questi servent, nella speranza di ottenere sufficienti informazioni per operare la scelta definitiva. Il messaggio include la chiave pubblica associata alla sessione di votazione. La chiave verrà utilizzata dai votanti per crittografare i messaggi di risposta, proteggendo quindi la riservatezza della loro identità e del voto in transito.

A questo punto *p* si trova a disposizione un insieme di giudizi da vagliare e da scremare: un problema da affrontare è quello di eliminare i voti sospetti di appartenere a delle cricche: si vuole evitare infatti che un utente attivi un grande numero di nodi sincronizzati per riuscire ad attribursi maggior peso nelle votazioni, spedendo un gran numero di voti. Così, una volta eliminati i voti non integri rispetto alla verifica delle firme digitali, si raggruppano sulla base della loro provenienza (riducendo l'effetto di possibili cricche) usando come riferimento topologico l'indirizzo IP. L'ipotesi è che sia difficile riuscire a connettersi alla rete Gnutella da un gran numero di indirizzi molto diversi tra loro.

Infine, alcuni di questi votanti, vengono contattati direttamente per chiedere conferma del voto. Per ogni votante selezionato viene generata una connessione diretta e tramite questa una richiesta `TrueVote` riportante i voti ricevuti, aspettandosi un messaggio di conferma `TrueVote Reply`. Questa procedura serve a verificare l'autenticità dell'indirizzo IP dichiarato dal votante.

Una connessione di questo genere, che non può certo andare a buon fine qualora il votante sia protetto da un firewall, può sfruttare il meccanismo del Push, richiedendo che il votante stesso si connetta a p dichiarando il proprio `ServentID`. Quello che si ottiene però è un certificato di minor qualità, essendo possibile, come precedentemente affermato, generare delle connessioni da indirizzi inventati. Un possibile scenario che esemplifichi questo attacco prevede un servent che voti dichiarando un indirizzo IP fasullo, protetto da firewall. All'arrivo del `Push`, questi genera una connessione da origine anonima per rispondere al messaggio `TrueVote`.

Notare che niente impedisce a servent scorretti di eliminare completamente dalla rete i voti in transito, cosa che del resto avrebbero potuto fare degli stessi messaggi `QueryHit`. Anche volendo non potrebbero tuttavia gettare voti selettivamente, così da falsare faziosamente le votazioni, dal momento che il contenuto dei messaggi `PollHit` è leggibile soltanto dal ricevente, essendo crittato con la sua chiave pubblica. Dopo aver verificato la correttezza dei voti, p può infine scegliere l'offerente che ha una miglior reputazione e quindi può essere ritenuto più affidabile.

Possono essere usati criteri differenti per decidere come effettuare la scelta, per esempio può essere deciso di scegliere l'offerente col più alto numero di voti positivi, oppure tra quelli che non hanno riportato voti negativi scegliere quello col maggior numero di voti positivi, oppure ancora esigere una soglia minima di voti per poi scegliere il miglior rapporto tra voti positivi e totali.

Prima di cominciare il trasferimento vero e proprio, p inizia la procedura `Challenge Response`, come illustrato in Figura 3.1(b), per verificare che l'identità dell'offerente corrisponda a quella estratta dal messaggio `QueryHit`. Questa procedura previene l'impersonificazione dei servent, in quanto solo il servent legittimo potrà rispondere correttamente al "challenge". La procedura si articola in questi punti:

- Il nodo p genera una stringa casuale di caratteri.
- p critta la stringa con la chiave pubblica del servent s al quale si vuole connettere
- p si connette a s tramite protocollo HTTP, e spedisce la stringa crittata.
- Il servent s, unico possessore della chiave privata associata a quella pubblica utilizzata per generare il `ServentID`, decritta il messaggio, lo firma con la sua chiave privata, e lo restituisce a p.
- p verifica l'identità del servent mediante controllo della firma digitale.
- Se la verifica va a buon fine p comincia la transazione, altrimenti chiude la connessione.

Terminata l'operazione di download deve essere dato un giudizio alla transazione sulla base della riuscita del trasferimento, dell'integrità e della qualità della risorsa prelevata, attribuendo un voto positivo, uno zero o un voto negativo al servent. Qualora la transazione non dovesse nemmeno terminare, o se il file dovesse risultare infetto da virus, l'utente non verrebbe nemmeno interpellato e verrebbe attribuito un giudizio negativo. Perchè sia dato un giudizio positivo serve invece l'intervento dell'utente, che deve valutare la qualità della risorsa e verificare che corrisponda alle

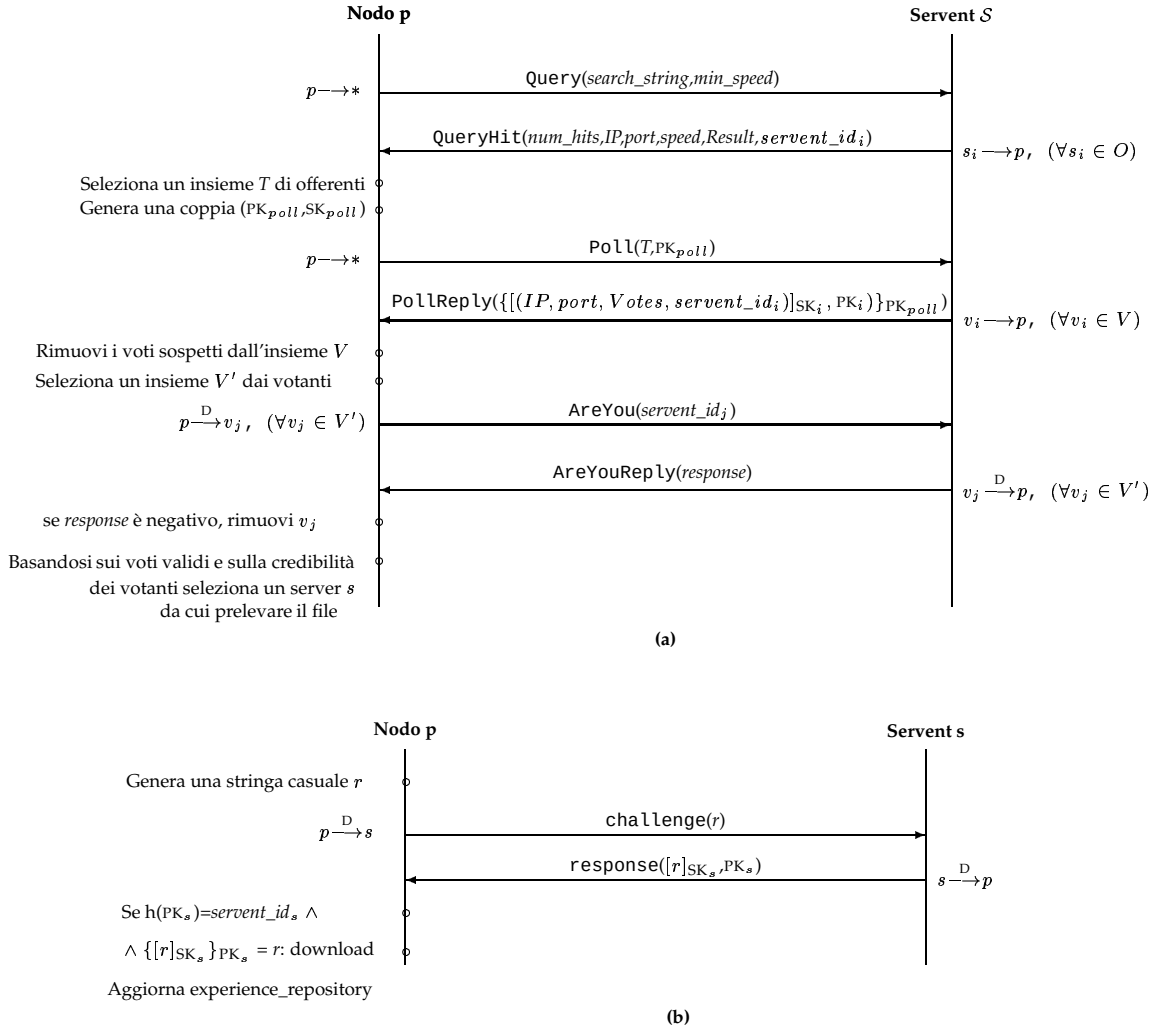


Figura 3.2: Sequenza di messaggi e di operazioni nel protocollo avanzato (a) e interazione con il servent selezionato (b)

sue aspettative. Questo giudizio viene registrato in un database dell'esperienza che contiene le informazioni relative ai servent con i quali sono state tentate delle transazioni. Queste informazioni vengono usate sia per rispondere ai voti `Poll`, sia per effettuare la prima scrematura all'arrivo dei messaggi `QueryHit`.

3.2.2 Protocollo avanzato

Il protocollo avanzato prevede che i votanti dichiarino la propria identità, in modo da poter valutare la loro integrità morale e attribuire un peso ai voti sulla base della loro credibilità. In Figura 3.2 viene illustrata la procedura completa: come per il caso base, dopo aver ricevuto i messaggi `QueryHit`, ed estratto da questi un sottoinsieme di servent che ragionevolmente possano soddisfare la richiesta (si può fare uso dell'esperienza acquisita per identificare i servent migliori), viene generato un messaggio di `Poll` per indagare la loro reputazione. [2]

Un servent che ricevendo questo messaggio voglia rispondere non deve far altro che generare un messaggio `PollHit` nel quale, a differenza del caso base, deve indicare il proprio `ServentID`.

[2] In alcuni casi può non essere necessario procedere con la votazione, essendoci già sufficienti garanzie ed informazioni per operare una scelta definitiva.

Il compito del votante è di esprimere un giudizio sui servent indicati nel messaggio `Poll` con i quali siano state portate a termine delle transazioni e di cui si abbia sufficiente informazione per poter esprimere dei giudizi.

Il carico del messaggio, composto dai campi indirizzo IP, porta del votante, elenco dei giudizi e chiave pubblica del votante, viene crittato con la chiave pubblica di p, estratta dal messaggio `Poll` relativo. Il messaggio comprende anche la firma digitale del carico per verificarne l'integrità. Crittare la chiave pubblica fornisce una garanzia di anonimato, non essendo infatti possibile risalire all'identità del votante se non da parte di chi indice le elezioni. Il fatto che i giudizi e i riferimenti assoluti del votante siano crittati protegge la loro riservatezza e impedisce la modifica selettiva dei valori. Inoltre la firma digitale garantisce l'autenticità della votazione, essendo generabile soltanto da chi conosce la chiave privata corrispondente al `ServentID`.

Ottenute le risposte, ognuna delle quali contiene un voto per ogni servent richiesto, si tratta di effettuare una scrematura per eliminare tutti i voti sospetti. Successivamente vengono selezionati e raggruppati (o addirittura eliminati) tutti i voti che che arrivano dallo stesso gruppo di IP, dando il sospetto di essere generati da una cricca.

Diventa importante per il protocollo garantire che l'indirizzo dichiarato sia quello vero, quindi questo viene controllato mediante un messaggio diretto di tipo `AreYou`: il messaggio contiene il `ServentID` associato all'indirizzo IP e chi se ne vede recapitare uno risponde `True` o `False` a seconda dei casi. Si noti che a differenza della versione Base non occorre verificare il voto, essendo stato autenticato da una firma digitale. È sufficiente infatti verificare l'identità del votante.

A questo punto p ha l'opportunità di sfruttare le conoscenze acquisite nelle precedenti votazioni, avendo a disposizione un metro di giudizio sulla credibilità dei votanti. Infatti, un votante che esprime un giudizio che finisce per essere concorde a quello verificato da p, subisce un'innalzamento nella scala della credibilità e alla votazione successiva verrà preso in maggior considerazione. Viceversa accade per le discordanze.

Mentre nel protocollo base ogni votante gode della stessa considerazione (una volta eliminati i casi sospetti), nel caso avanzato interviene un coefficiente di adattamento proporzionale alla credibilità acquisita dai votanti. Ogni servent infatti tiene traccia del numero di concordanze e di discordanze del proprio giudizio finale su quelle proposte da ogni votante.[3]

Effettuata la valutazione dei voti, i parametri da tenere in considerazione per la scelta finale della risorsa da prelevare sono: velocità dichiarata di connessione (eventualmente si considera anche la velocità dinamica calcolata come media di trasferimento degli ultimi dieci file, come indicato da BearShare), la reputazione del server ottenuta da precedenti interazioni e infine la reputazione indicata dai votanti.

Come per il protocollo base anche in questo caso, prima dell'effettivo trasferimento dei dati, viene attuato uno scambio `Challenge/Response`, in modo da verificare l'identità del server.

3.2.3 Reputazione e Credibilità

Nella descrizione del protocollo è stato accennato al fatto che ogni servent debba tenere traccia delle credibilità e della reputazione di nodi con i quali viene in contatto, ma non si sono specificati i dettagli. Ci sono diversi modi di affrontare il problema di gestire, mantenere e tradurre in termini

[3] Queste informazioni risultano interessanti anche nel caso qualche votante risulti fortemente negativo, perchè si potrebbe tendere, in questo caso, ad agire nel modo contrario a quello segnalato.

di voti le informazioni derivanti dalle interazionii. In questa sezione si illustra il metodo adottato nel protocollo attuale. Il metodo è stato scelto con criteri di semplicità ed economia (per ogni servent vengono mantenuti uno o due record di piccole dimensioni).

Ogni servent gestisce le reputazioni associando ad ogni altro nodo una quaterna di valori:

```
(ServentID, n.plus, n.minus, timestamp)
```

`n.plus` rappresenta il numero di interazioni positive, mentre `n.minus` il numero di quelle negative. Un'interazione negativa è riconoscibile per il fatto di non essere giunta a compimento per cause legate al server (connessione interrotta, server irraggiungibile), oppure per il fatto di aver prodotto un materiale inadatto alle aspettative, corrotto o addirittura dannoso (presenza di virus, Trojan, denial of service o worm). Un'interazione è buona qualora il file scaricato risponda alle aspettative.

Il `timestamp` è il riferimento temporale all'ultimo accesso al record: il database viene infatti mantenuto entro certi limiti di spazio eliminando i record meno utili.

Per avere la possibilità di estrarre dei giudizi più ragionevoli sarebbe interessante tener traccia di tutte le informazioni legate ai download ma risulterebbe piuttosto difficile garantire l'anonimato della rete, poiché sarebbe possibile ricostruire, avendo accesso alla storia di molti servent, un insieme di transazione riconducibili direttamente a servent specifici.

La tabella delle reputazioni viene aggiornata ad ogni interazione e riflette l'esperienza diretta di chi la mantiene: per questo deve venir presa in gran considerazione dalle specifiche di scelta delle risorse e farla prevalere, in caso di parità, alle reputazioni indicate dai votanti.

Un problema da affrontare è la rappresentazione dei voti, che in letteratura è stata affrontata in diversi modi. eBay, per esempio, usa un insieme di tre voti possibili: -1, 0, 1 per rappresentare rispettivamente un giudizio cattivo, uno medio e uno buono. Per gli scopi di questo protocollo non è necessario considerare il caso medio, dal momento che può essere sostituito dall'assenza di voto. Così i voti proposti sono solo due: 0 per rappresentare un giudizio negativo, 1 per rappresentare quello positivo.

Per ottenere il voto a partire dai valori presenti nel database della reputazione si è scelto di lasciare libera iniziativa alle implementazioni, qui ci si limita a suggerire due possibili approcci: stabilita una soglia minima di interazioni sotto la quale non si esprimono giudizi, si restituisce 1 qualora il rapporto tra il numero di interazioni positive ($plus$) e totali ($plus + minus$) risulti maggiore di un valore fissato, che potrebbe essere $0,8$. Un altro modo possibile consiste nel restituire 0 ogni qual volta il numero di interazioni negative sia superiore ad una certa soglia e restituire 1 se il numero di rapporti positivi sia maggiore di un'altra.

Qualora un'interazione risultasse fortemente dannosa (questo potrebbe accadere nel caso in cui si dovesse riconoscere un virus o un Trojan), allora si potrebbe decidere di aumentare il numero di interazioni negative di una quantità superiore ad uno, in modo da aver la certezza di evitare di contattare ancora lo stesso servent, rivelatosi oltre ogni ragionevole aspettativa pericoloso.

Per quanto riguarda il database della credibilità è necessario mantenere le seguenti informazioni per ogni votante:

```
(ServentID, n.concordanze, n.discordanze, timestamp)
```

Per `n.concordanze` e `n.discordanze` si intende il numero di volte nelle quali il giudizio finale rispettivamente concorda o discorda con il votante. Un modo semplice per gestirlo può essere di incrementare il contatore di uno o dell'altro campo ogni volta che si abbia modo di verificare una

risorsa scaricata e premiare coloro che hanno espresso una votazione concorde con il giudizio verificato dall'analisi della transazione e dalla risorsa e, viceversa, punire chi era in disaccordo.

3.2.4 Riconoscere e rimuovere i voti sospetti

I messaggi `PollHit` devono essere verificati per evitare che servent che vogliano falsificare le votazioni riescano a generare un numero sufficiente di voti falsi. Un possibile scenario prevedere che un servent possa generare molti voti falsi o cancellarne altri in modo da alterare il giudizio del ricevente.

L'idea che è stata adottata per ridurre il rischio legato alla creazione di voti falsi è cercare di riconoscere i voti sospettati di provenire da un solo servent considerandoli quindi un singolo voto. Per fare questo sono state utilizzate delle tecniche di clustering. La scelta del metodo di clustering deve tendere a rendere il più possibile complessa la produzone di voti, da parte di un solo servent, provenienti da cluster differenti. Nel contempo la divisione a cluster non deve raggruppare voti espressi effettivamente da nodi differenti. A supporto del metodo di clustering, alcuni votanti vengono contattati direttamente, così da verificare l'appartenenza ai cluster a cui votanti dichiarano di appartenere.

Ci sono diversi modi per raggruppare i voti in cluster ma quello più semplice prevede il controllo dell'indirizzo IP, raggruppando indirizzi che appartengono alla stessa `net_id`. Una soluzione più robusta, utilizzata da strumenti come NetGeo e IP2LL, raggruppa gli indirizzi accedendo agli archivi Whois, in modo da ottenere i blocchi che includono un determinato indirizzo IP. Questa soluzione pone il problema, per esempio, di raggruppare in unico voto tutti i voti provenienti da nodi dell'università di Milano, o di tutta la rete AOL.

Una variante interessante a questo processo, che eviterebbe completamente l'esigenza di generare dei messaggi di controllo diretto dell'IP, sfrutta la natura specifica di Gnutella, che prevede per ogni nodo un certo numero di connessioni. L'idea è quella di raggruppare i voti per connessione e infine mediare i voti. In questo modo, un nodo che volesse creare un numero alto di servent, ognuno dotato di propria chiave persistente e di una politica comune di voto, difficilmente potrebbe comparire significativamente nella maggior parte delle connessioni. Infatti l'orizzonte di visibilità è molto lontano dalla visione completa della rete, ed è difficile prevedere a quali nodi si voglia connettere un particolare servent.

3.3 Valutazioni sulla sicurezza di P2PRep

Il protocollo P2PRep nasce con l'intento di migliorare la sicurezza complessiva degli ambienti peer-to-peer, con particolare riferimento a Gnutella, ambiente privilegiato per la sua apertura alle novità. Il protocollo P2PRep cerca inoltre di risolvere una serie di problemi ben conosciuti, strettamente legati agli ambienti peer-to-peer e agli algoritmi di votazioni distribuite. Nell'analisi che segue si definisce una tassonomia degli attacchi possibili a questo genere di protocolli. Si mostrano poi le tecniche attuabili per eliminare (o perlomeno a ridurre) il loro impatto. Lo scenario degli attacchi prevede che *Alice* sia un utente Gnutella alla ricerca di un determinato file, *Bob* un utente che possiede quella risorsa, *Carlo* un utente protetto da un firewall (anch'egli possessore del file) ed infine *Davide*, un utente malvagio.

3.3.1 Distribuzione di informazioni contraffatte

Questo attacco prevede che Davide risponda alle ricerche di Alice cercando di spedire una risorsa manomessa, avente lo stesso nome e la stessa dimensione del file cercato. Il file spedito potrebbe essere un Trojan horse o contenere un virus (come ad esempio il worm Gnutella accennato precedentemente). Attualmente questo tipo di attacco è molto praticato, essendo piuttosto facile da mettere in atto: nella sua versione più semplice non serve nemmeno operare delle modifiche al software Gnutella. Una versione più raffinata dell'attacco invece potrebbe articolarsi come segue.

1. **Fase di raccolta dati**: in questa prima fase vengono raccolti tutti i messaggi `QueryHit` che passano per la rete, operazione che richiede la modifica di un client. L'Appendice A si mostra l'implementazione di un client utilizzabile per raccogliere questi dati.
2. **Fase di Ordinamento**: la lista dei file più offerti rappresenta una statistica importante della rete Gnutella e del suo traffico. Non necessariamente i file più offerti sono quelli più scaricati, tuttavia esiste certamente una correlazione con quanto viene ricercato. Di ognuno di questi file si conosce il nome, la dimensione e la frequenza con la quale viene offerto.
3. **Preparazione dello scenario**: scelto un insieme di file da riprodurre, che possano essere ritornati in risposta a ricerche con alta probabilità di essere ricevute, si generano dei Trojan o dei Virus della lunghezza e con il nome specificato.
4. **Attacco**: per l'attacco vero e proprio basta utilizzare un client Gnutella qualunque al quale verrà specificata la directory di condivisione preparata nel punto precedente. Ogni qual volta un servent ricerchi questi file, che ricordiamo essere stati scelti in modo da essere richiesti con una certa frequenza, vengono automaticamente offerti.

Un attacco come quello descritto sarebbe oltresì in grado di mettere in crisi sistemi di download parallelo del tipo implementato da FastTrack, perchè renderebbe inutilizzabile un intero file anche se solo una piccola parte di questo venisse prelevata, da un server che ne ha contraffatto la copia.

Il protocollo `P2PRep` è tuttavia in grado di limitare questo tipo di attacco, poichè appena Alice dovesse accorgersi della contraffazione di una risorsa offerta da Davide, ne modificherebbe la reputazione. Alice eviterebbe quindi di effettuare con Davide altre operazioni e sconsiglierebbe contatti con lui in risposta a `Poll`. Lo sforzo fatto da Davide per acquisire una buona reputazione verrebbe vanificato, richiedogli di cambiare identificativo per sfuggire a questi nuovi voti negativi. Si ridurrebbero quindi in modo significativo le sue speranze di far propagare il materiale contraffatto.

3.3.2 Man in the middle

L'attacco Man in the middle, illustrato in Figura 3.3, prevede che nella comunicazione tra Alice e Bob possa intromettersi una terza figura (Davide), capace di influenzare il traffico così da trarne qualche tipo di beneficio. Un attacco potrebbe svolgersi come segue:

1. Alice inoltra una ricerca e Bob risponde con un messaggio `QueryHit`.
2. Davide intercetta la risposta e sostituisce l'indirizzo IP di Bob con il proprio.
3. Alice riceve la risposta.

Figura 3.3: Man in the middle

4. Alice decide di scaricare la risorsa da Davide.
5. Davide scarica la risorsa originale da Bob e la passa ad Alice dopo averla infettata.

Una possibile variante che utilizzi il metodo del `Push` è la seguente:

1. Alice genera una ricerca. Risponde Carlo, protetto da un Firewall.
2. Alice, per recuperare la risorsa, manda un messaggio `Push`.
3. Davide intercetta il messaggio e risponde con il proprio indirizzo IP.
4. Carlo riceve il `Push` e spedisce il messaggio a Davide.
5. Davide si connette ad Alice e inoltra il file, dopo averlo infettato.

Entrambi questi attacchi sono prevenuti dal protocollo `P2PRep` grazie alla tecnica *Challenge-Response* effettuata subito prima dello scaricamento effettivo dei file. Per riuscire ad impersonificare Bob o Carlo, Davide dovrebbe conoscere la loro chiave privata, oppure essere in grado di generare una chiave pubblica che abbia lo stesso digest della loro. Tali operazioni sono tuttavia impraticabili in tempi ragionevoli.

3.3.3 Tradimento

Un utente potrebbe acquisire una certa reputazione, comportandosi bene per un certo periodo, per poi cambiare repentinamente strategia e cominciare a distribuire file infetti o corrotti. Il protocollo è in grado di adattarsi a questo genere di attacco richiedendo un numero basso di comportamenti scorretti per ridurre al minimo la reputazione. Questo accorgimento comporterebbe un dispendio eccessivo di energie da parte dell'attaccante, che vedrebbe vanificato un lungo periodo di costruzione di una buona reputazione in poco tempo.

3.3.4 Accerchiamento

Davide potrebbe voler accerchiare Alice frapponendo un'istanza del suo client per ogni connessione di Alice, in modo da non permetterle di raggiungere la rete senza attraversarlo. In Figura 3.4

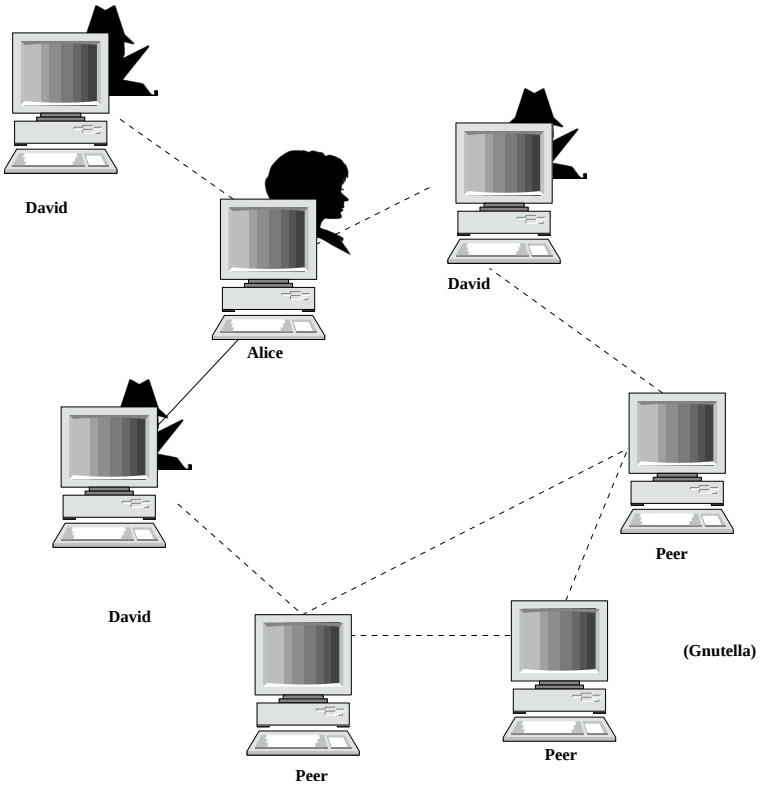


Figura 3.4: Accerchiamento di un nodo

si vede illustrato il caso in cui Alice, con tre connessioni disponibili, non abbia modo di raggiungere la rete senza passare per Davide. In questo modo egli avrebbe un controllo completo dei messaggi che la raggiungono e potrebbe cancellarli selettivamente o modificarli per indurla a scaricare quello che desidera.

Una possibile soluzione prevede che le connessioni debbano avvenire verso indirizzi che rendano uniforme e massima la distanza tra le connessioni, in una metrica che consideri la distanza uguale alla differenza algebrica degli indirizzi IP. Il protocollo P2PRep prevede invece che le connessioni avvengano, se possibile, con servent di cui si stima la reputazione. A Davide riuscirebbe pertanto difficile creare diverse istanze tutte distinte e di alta reputazione, tali da convincere Alice a connettersi loro. Inoltre per Davide non sarebbe comunque facile agire selettivamente sui messaggi `PollHit`, essendo questi crittati con la chiave pubblica di Alice.

3.3.5 Recupero di informazione selettiva sui servent

Gnutella, a differenza di altri ambienti peer-to-peer come Napster, non mette a disposizione alcuna primitiva per visualizzare l'intero contenuto di una directory condivisa. Tuttavia non esistono particolari precauzioni per evitare che questo accada. In particolare, se Davide volesse conoscere la lista completa dei file condivisi da Alice, potrebbe inoltrare delle ricerche con tutte le più ragionevoli combinazioni di tre lettere (misura minima di ricerca presa in considerazione dalle implementazioni) e selezionare dalle risposte quelle che appartengono ad Alice. Il nostro protocollo non fornisce una soluzione a questo tipo di problema anche se sono possibili alcune modifiche per ridurne l'impatto.

Una possibile soluzione potrebbe essere quella di limitare il numero di risposte date in ogni messaggio `QueryHit`, selezionando casualmente pochi file alla volta dalla lista di tutti quelli che

soddisfano la ricerca. In questo modo si richiede a Davide un impegno prolungato per avere una lista ragionevolmente completa.

Gnutella non nasce, del resto, con strette esigenze di anonimato, tuttavia esistono diverse proposte che tendono alla sua introduzione. Una di queste permette di effettuare connessioni ad un servent tramite una catena di altri, crittando a cipolla i messaggi[69]. L'idea è che un nodo A che voglia mandare un messaggio ad un nodo D passando per B e C, come prima operazione si procuri le chiavi pubbliche di tutti questi nodi. Poi critta il messaggio con la chiave pubblica di D e lo incapsula in nuovo messaggio, indirizzato a D. Tutto questo messaggio viene crittato con la chiave pubblica di C, incapsulato in un altro messaggio questa volta crittato con la chiave di C. Così anche per B. Infine, quest'unico messaggio che incapsula, come una cipolla, tutta la catena di passaggi voluti, viene spedito a B. Questi è l'unico in grado di aprire il contenuto, composto da un messaggio (ch'egli non è in grado di intendere) e dall'indirizzo di C. B manda quindi il contenuto a C, che può decrittare la busta e mandarne il contenuto a D, il quale può finalmente leggere il messaggio originale. Notare che nessun nodo della catena, a parte A, conosce le parti coinvolte nella comunicazione. Svantaggio di questo metodo è che richiede un gran numero di passaggi e il propagarsi in rete di messaggi di grosse dimensioni.

3.3.6 Cricca

Controllare gli attacchi di un solo malintenzionato è decisamente meno impegnativo di prevedere il comportamento sincronizzato di un insieme di attaccanti. Per cricca si intende un insieme di nodi, distinti tra loro, che perseguono in modo organizzato un determinato fine. È evidente che i protocolli che sfruttano le votazioni e si fidano della maggioranza conducono verso la scelta ottimle fintanto che la maggioranza voti onestamente e con saggezza. Se una cricca dovesse superare una quantità critica, tutto il sistema crollerebbe, ma lo sforzo richiesto per ottenere tale destabilizzazione sarebbe immenso. Un attacco plausibile prevede invece un massimo di un centinaio di nodi. Questi difficilmente riuscirebbero a distribuirsi in modo uniforme nelle metriche adottate da P2PRep sugli indirizzi e sulle connessioni. Il controllo sulle cricche avviene proprio in questo modo: da una parte si tende a diversificare le connessioni e dall'altra a raggruppare i voti.

3.4 Possibili estensioni di P2PRep

Il protocollo P2PRep prevede che la reputazione sia associata ai servent, i quali sono gli unici garanti delle risorse che mettono a disposizione. Una variante possibile del protocollo, per certi versi complementare denominata XP2PRep, prevede che la reputazione venga attribuita alle risorse, in contrapposizione ai servent. Intuitivamente, ad ogni risorsa viene associato un digest calcolato dal contenuto del file mediante una funzione di hash sicura. Quando un servent scarica una risorsa, identificata dal suo digest, tiene traccia del fatto di esserne stato soddisfatto o meno. Questo giudizio viene mantenuto in una tabella simile a quella delle reputazioni, definita nel protocollo P2PRep. I messaggi `Poll`, invece di chiedere informazioni sui servent, hanno il compito di raccogliere le opinioni della comunità sulle risorse. Per fare ciò è necessario che i messaggi `QueryHit`, oltre al nome e ai riferimenti delle risorse, contengano anche il loro digest. Il funzionamento di questo protocollo, per il resto, è pressoché uguale al protocollo P2PRep.

Conseguenza di questo diverso modo di intendere il protocollo è la possibilità di garantire in modo sistematico l'integrità dei file e la correttezza del suo trasferimento. Interessante notare che il garante di una risorsa non coincide più con il suo possessore, pertanto i piccoli servent possono redistribuire risorse diminuendo il carico dei servent più grandi. Infatti è più che plausibile pensare che i grandi distributori di file, che statistiche dimostrano essere un numero esiguo di server, con l'implementazione del protocollo P2PRep, possano ottenere un alto valore di reputazione. A questa reputazione corrisponderebbe un ulteriore aumento del loro carico.

Del protocollo XP2PRep esiste un'implementazione [58] che permette di verificarne il corretto funzionamento in una reale rete Gnutella. Questo software permette di istanziare dei client in grado di gestire i nuovi messaggi `XPoll` e `XPollHit`, e di rispondere correttamente ai normali messaggi `Query` e `QueryHit`. La configurazione del comportamento di ogni client è completamente definita da un file di configurazione, nel quale è possibile specificare quali connessione stabilire, quali file offrire e quali file richiedere.

Capitolo 4

Implementazione di P2PRep.

Tu non lo udisti e il tempo passava
con le stagioni al passo di java
ed arrivasti a varcar la frontiera
in un bel giorno di primavera.

Fabrizio De André

Anche se esistono diverse implementazioni del protocollo Gnutella, tutte quante tendono a mantenere la stessa struttura architetturale. In questo capitolo vengono evidenziate le parti salienti della struttura di Gnutella tramite tecniche di ingegneria del software. Si utilizzano infatti schemi standard UML [12].

L'obiettivo è quello di fornire i dettagli necessari all'implementazione del protocollo P2PRep. Si ritiene necessario mantenere una stretta interoperabilità tra i protocolli e verranno illustrati dei metodi per permettere ai nuovi client di utilizzare la rete Gnutella preesistente. Siccome il protocollo P2PRep estende il protocollo Gnutella, si pensa che sia opportuno implementarlo partendo da una implementazione esistente di un client Gnutella. L'attuale implementazione è stata scritta in Java e utilizza delle API Gnutella chiamate JTella, opportunamente modificate. In questo capitolo si illustrano tutte le modifiche necessarie ad un normale client Gnutella e in particolare verrà illustrato come incapsulare i messaggi `Poll` e `PollHit` all'interno di normali messaggi `Query` e `QueryHit`. Infine si mostra come implementare il protocollo XP2PRep.

4.1 Architettura del sistema

In questa sezione si presentano tutti gli schemi necessari per illustrare l'architettura di un sistema P2PRep. Si parte con i casi d'uso, che ne definiscono le funzionalità. Segue una visione d'insieme che mostra le parti che compongono l'architettura. Di ognuna viene esplicitato il funzionamento. Successivamente si presenta uno schema delle classi che rappresenta tutte le interazioni salienti e tutti i metodi principali delle classi utili per implementare il sistema. Per ogni caso d'uso si definiscono poi i diagrammi di sequenza, utili per comprendere l'evolversi di ogni funzionali-

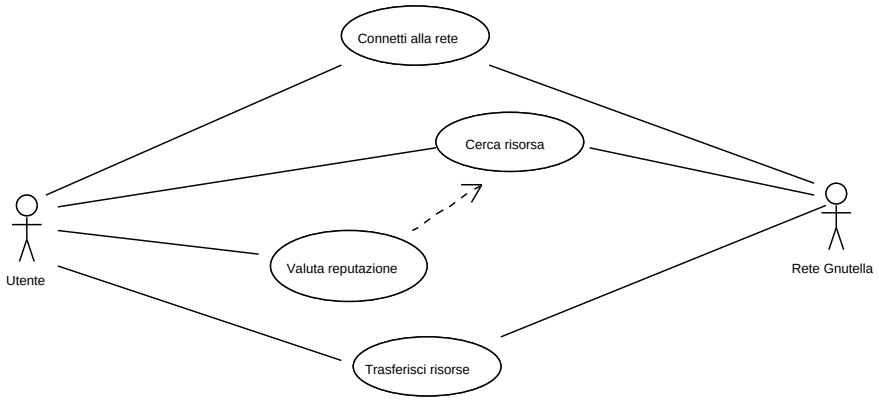


Figura 4.1: Caso d'uso per l'utente

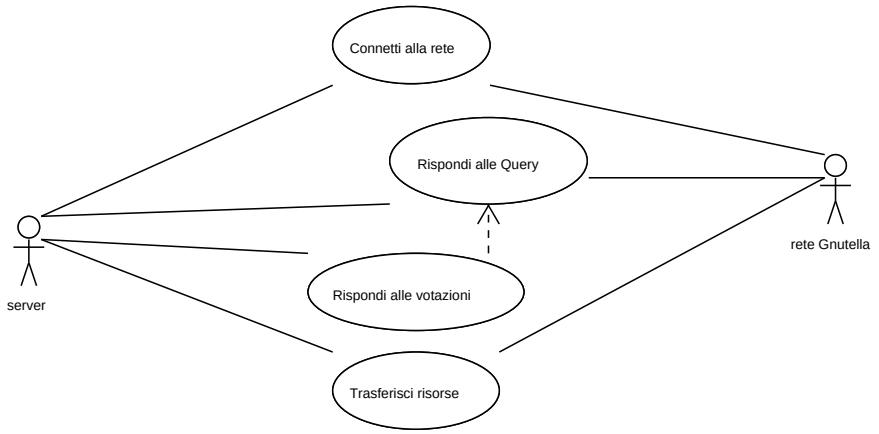


Figura 4.2: Caso d'uso per il server

tà. Infine si descrive la sintassi del file di configurazione del prototipo, utile per effettuare delle sperimentazioni sul protocollo.

4.1.1 Casi d'uso

Sono stati evidenziati due casi d'uso che rispecchiano i due aspetti di una sessione di un servent P2PRep: il lato client (Figura 4.1) e quello server (Figura 4.2).

Client: L'utente, dopo essersi collegato alla rete Gnutella, genera una "query" per la ricerca di risorse. Qualora le trovasse può utilizzare lo stesso meccanismo per valutare la qualità della sorgente. Successivamente può procedere con il trasferimento dei file.

Server: Ogni server mette a disposizione delle risorse, e nel caso qualche client le richieda, deve poter rispondere fornendo i dettagli utili per il trasferimento. Deve inoltre saper fornire una valutazione dei servent di sua conoscenza e permettere, a chi ne faccia richiesta, di scaricare i file condivisi.

4.1.2 Visione d'insieme

Il diagramma architetturale del progetto, riportato in Figura 4.3, evidenzia le componenti fondamentali del progetto.

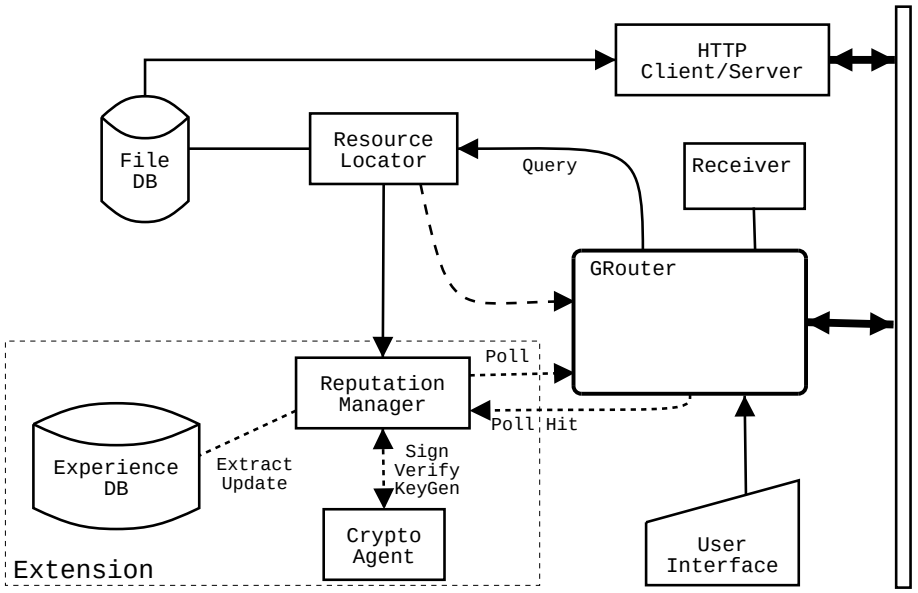


Figura 4.3: Architettura Reputella

La connessione alla rete, la gestione dei messaggi e il loro indirizzamento viene gestito dal `GRouter`, componente centrale del sistema. Il *Receiver* si occupa della ricezione dei messaggi indirizzati al servent stesso e riconosce la presenza di eventuali messaggi estesi. L'analisi del file di configurazione, l'interazione con l'utente e la compilazione dei `Log` sono assegnati alla componente `UserInterface`. Le risorse e le loro specifiche sono detenute nel `FileDatabase`, gestito dal `ResourceLocator`. I messaggi diretti e lo scaricamento del file sono gestiti da due componenti separate, chiamate `HTTP Client/Server`. È utile separare le estensioni dal protocollo originale per riuscire ad ottenere un'implementazione funzionante partendo da software preesistente, in modo da minimizzare il tempo necessario alla creazione di un prototipo. L'estensione del protocollo di base è gestita dalle tre componenti `ReputationManager`, `CryptoAgent` e `ExperienceDB`. La gestione delle reputazioni è affidata al `ReputationManager`, che interagendo col `CryptoAgent` e con il `ExperienceDB` è in grado di fornire informazioni e valutazioni relative alle interazioni con altri servent effettuate nel corso della sua storia.

Uno degli aspetti critici di questo progetto, che richiede particolare attenzione, è il riconoscimento dei nuovi pacchetti e la loro gestione. Queste operazioni vengono implementate nel `Receiver`. Il diagramma degli stati in Figura 4.4 schematizza il suo comportamento.

Ogni nuovo messaggio viene analizzato per controllarne il tipo e per sapere se necessita di una gestione estesa o normale. I messaggi estesi sono: `Poll` e `PollHit`. I messaggi normali sono `Query` e `QueryHit`. Ogni richiesta di risorse è definita da una stringa di ricerca che viene passata al `Resource Locator`, in grado di verificare l'eventuale presenza di file che le corrispondano. In questo caso verrà generato un messaggio di `QueryHit` indirizzato al servent che ha generato la richiesta.

Nel caso in cui il messaggio pervenuto sia invece di tipo `QueryHit`, allora significherebbe che è stata data risposta ad una `Query` precedentemente inoltrata nella rete. Raggiunto un numero sufficiente di risposte, ognuna delle quali in grado di identificare tramite `ServentID` il servent che l'ha generata, viene invocato il `Reputation Manager` per stabilire un ordine qualitativo basato sulla loro reputazione.

In caso di indecisione o di informazione insufficiente viene creato e spedito in broadcast un mes-

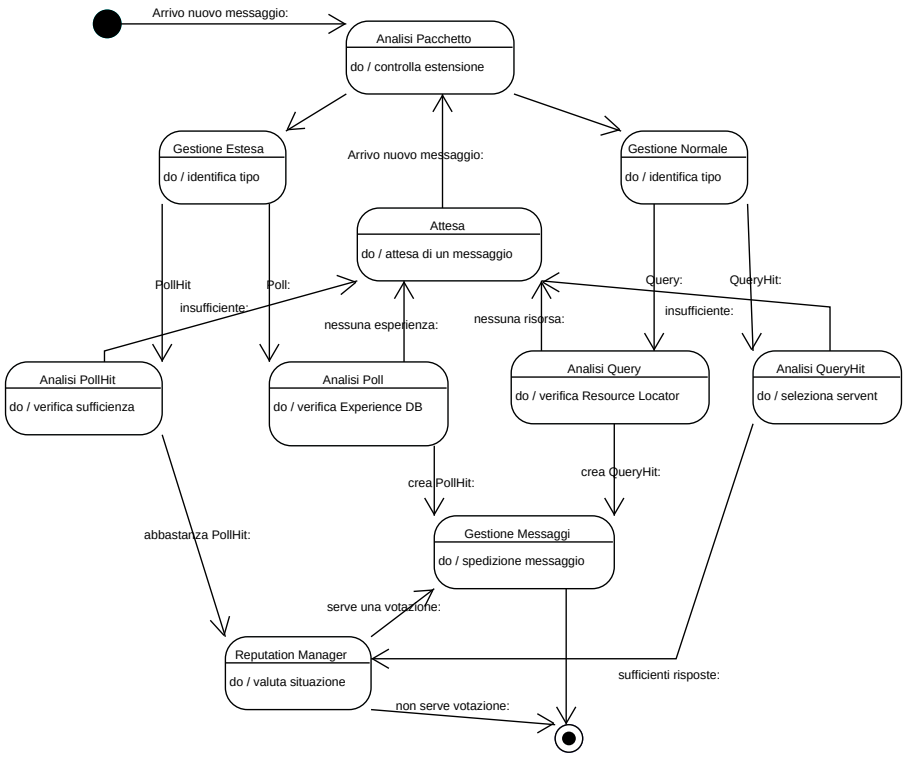


Figura 4.4: Diagramma degli stati della componente Receiver

saggio di `Poll` contenente gli identificativi dei *servent* di cui si vuole avere l'opinione della comunità. A loro arrivo questi vengono spacchettati dalla gestione estesa similmente a come si opera sulle `Query`, con la differenza che le risorse verranno cercate nel database dell'esperienza acquisita. La loro ricerca avviene da parte del *Reputation Manager*.

I pacchetti `Poll` contengono inoltre la chiave pubblica del servent che lo ha generato, dalla quale è possibile ricavare il suo identificativo tramite invocazione del `CryptoAgent`. L'arrivo di messaggi `PollHit` implica il fatto che sia stata indetta una votazione: dopo un tempo prestabilito o dopo l'arrivo di un numero sufficiente di messaggi di questo tipo, questi vengono analizzati per scegliere il servent dai quale scaricare infine la risorsa.

4.1.3 Schema delle classi

Lo schema UML delle classi proposto nella Figura 4.5 è composto essenzialmente da tre package: `Crypto` che gestisce tutte le funzioni crittografiche necessarie, `DirectDownload`, che si occupa della trasmissione dei dati diretta tramite protocollo HTTP e `Reputella` che definisce la struttura della parte principale del programma.

Il `GRrouter` inoltra e riceve messaggi, tutti derivanti dalla stessa interfaccia `Message`. `Poll` deriva da `Query`, mentre `PollHit` deriva da `QueryHit`, essendo loro estensioni naturali. `JTella` mette a disposizione l'interfaccia `MessageReceiverAdapter`, estesa da `XMessageReceiverAdapter`, per incorporare la gestione dei nuovi messaggi. `JTella` gestisce le connessioni tramite oggetti `GNUtellaConnection`, riferiti nella classe `GRouter`. Il database dei file e delle risorse è implementato dalla classe `FileDB`, i cui metodi di ricerca vengono invocati da `ResourceLocator`. I database dell'esperienza sono invece gestiti dalla classe serializzabile `DB` che riferisce, tramite

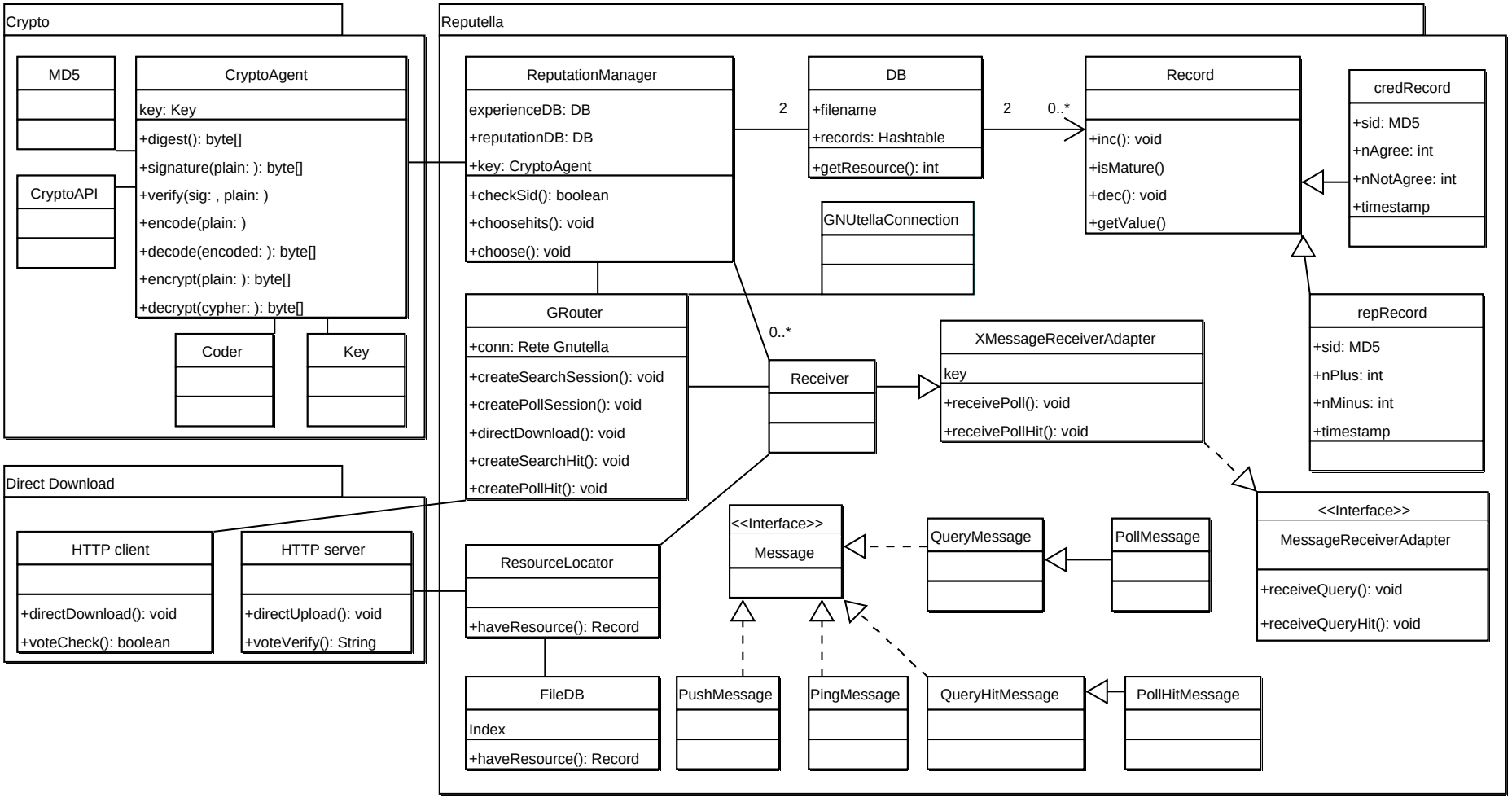


Figura 4.5: Diagramma delle classi

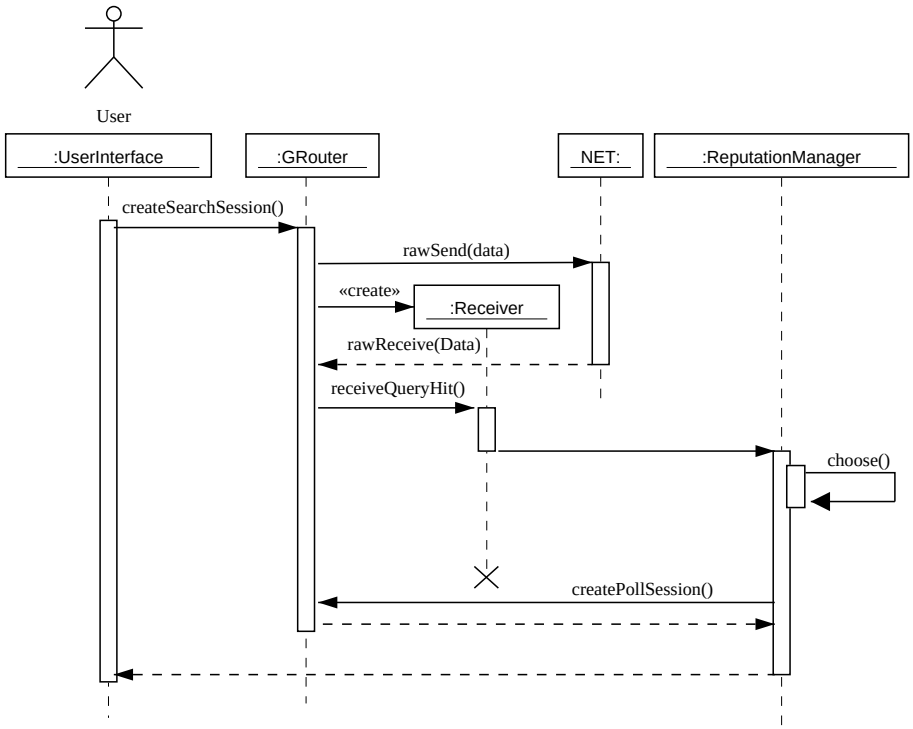


Figura 4.6: Diagramma di sequenza di una sessione Query

hashtable, i `Record`. Il `ReputationManager` istanzia due oggetti `DB`, uno per l'esperienza acquisita nelle transazioni che valuti la reputazione dei servent, l'altra per la credibilità dei votanti. `CryptoAgent` è una classe in grado di offrire staticamente tutte le funzioni crittografiche necessarie. Le chiavi sono rappresentate dalla classe `Key` e ogni oggetto `Receiver` ne deve istanziare una coppia diversa. Una coppia persistente viene invece mantenuta dal `CryptoAgent`.

4.1.4 Diagrammi di Sequenza

Il caso d'uso client prevede che ci sia la possibilità di effettuare delle ricerche (Figura 4.6) all'interno della rete come specificate dalla componente `User Interface` invocando il metodo di ricerca `createSearchSession(criteria)` dalla componente `GRouter`, il quale si occupa di spedirlo sulla rete. All'atto della creazione della ricerca viene istanziato un oggetto `Receiver` che si occuperà della ricezione dei messaggi in risposta alla suddetta `Query`. Quando arriva un messaggio `QueryHit` relativo alla ricerca, questo viene indirizzato alle componente ricevente che si occupa della gestione del pacchetto. Ottenuto un numero sufficiente di messaggi di questo genere, questi vengono inoltrati al `ReputationManager`, che può decidere di generare una sessione di votazione (Figura 4.7), invocando la generazione di un pacchetto `Poll`. Altrimenti li restituisce allo `UserManager`. Tramite il metodo `createPollSessioni()`, `Grouter` genera messaggi di `Poll`. Analogamente a quanto avveniva con i messaggi `Query`, viene associato a questa ricerca un oggetto di ricezione, che connesso all agente di crittografia, è in grado di ricevere di analizzare i messaggi di `PollHit`.

Dal lato server (Figura 4.8) l'arrivo di un messaggio `Query` viene intercettato dalla componente di ricezione, che verifica tramite invocazione del metodo `haveResource()` dell'oggetto `Resource Locator` la presenza della risorsa cercata. Analogamente viene gestito l'arrivo di messaggi `Poll` (Figura 4.9), che tuttavia, verranno riferiti ai database dell'esperienza (in contrapposizione al da-

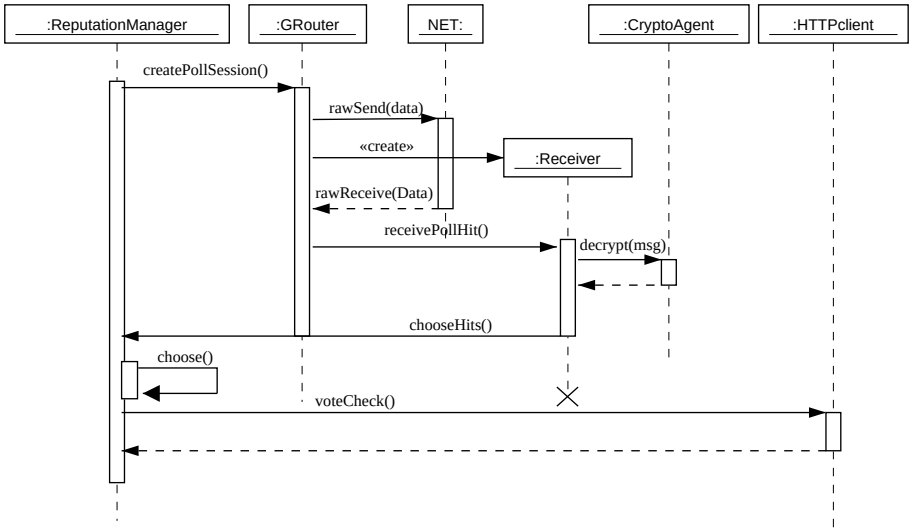


Figura 4.7: Diagramma di sequenza di una sessione Poll

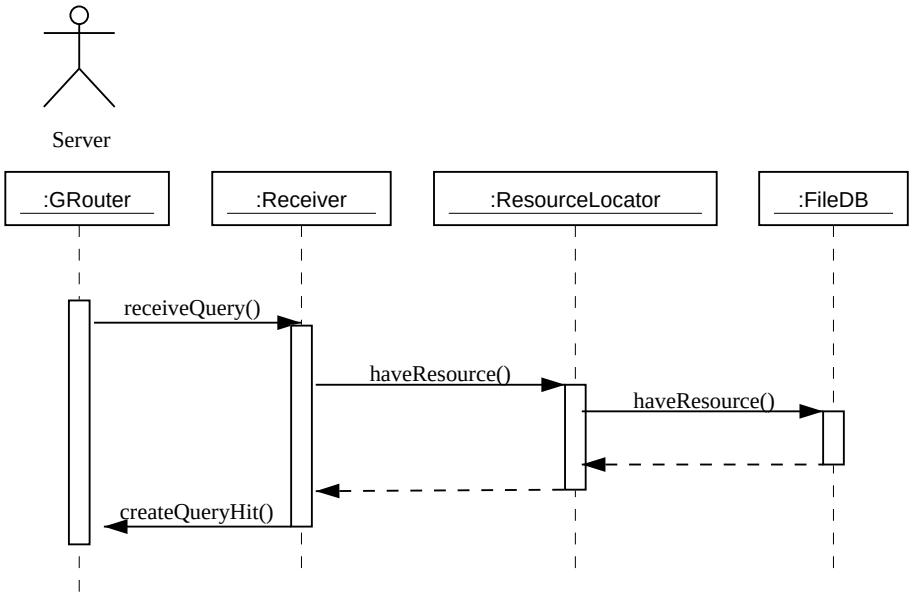


Figura 4.8: Diagramma di sequenza di una sessione QueryHit

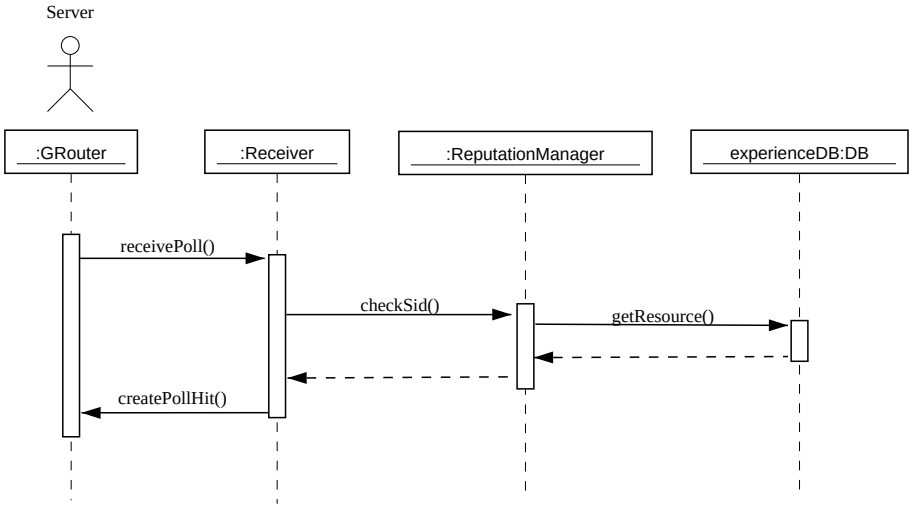


Figura 4.9: Diagramma di sequenza di una sessione PollHit

tabase FileDB utilizzato nelle ricerche). Le informazioni utili alla generazione dei pacchetti di risposta `PollHit` vengono ottenute invocando l'oggetto `ReputationManager`.

4.1.5 Considerazioni sui database dell'esperienza

Per gestire i database dell'esperienza, che comprendono reputazione e credibilità, esistono diverse soluzioni possibili. Di seguito ne viene proposta una. Ogni record delle tabelle dei database tiene traccia di quattro valori: un identificativo di servent, due contatori e un timestamp. I contatori servono a misurare il numero di interazioni positive e negative, che nel database della reputazione si traducono in giudizi sulle transazioni effettuate, mentre per quanto riguarda il database della credibilità riflettono le concordanze tra i giudizi sulle transazioni e quelle del votante.

Per riassumere, gli schemi sono i seguenti:

REPUTAZIONE : `(ServentID, n.plus, n.minus, timestamp)`

CREDIBILITÀ : `(ServentID, n.concordanze, n.discordanze, timestamp)`

La presenza del timestamp risulta necessaria per evitare il mantenimento di record vecchi ed inutilizzati: ogni volta che un record viene utilizzato, si aggiorna il timestamp. Un modo per mantenere i record è infatti quello di gestire la lista mediante tecnica *Not Frequently Used* [70] che prevede di associare all'insieme di record un indice ordinato per timestamp. Nel caso in cui venga raggiunto il limite prefissato di record e sia necessario aggiungerne altri, vengono eliminati i più vecchi.

La gestione di questi database non è eccessivamente dispendiosa in termini di spazio, infatti ogni record è lungo 28 bytes. Questo significa che con meno di 700Kb si possono trattare 25.000 servent, un numero più che sufficiente per descrivere tutti i nodi attivi della comunità.

4.1.6 File di configurazione

Il file di configurazione è composto da 5 parti: CONFIGURATION, FILES, REPUTATION, CREDIBILITY, CONNECTIONS e COMMANDS. Nella parte generale si specifica quale porta venga utilizzata per le connessioni, il numero massimo di connessioni in entrata e in uscita e il nome del file di log. Le connessioni sono definite, una per riga, da un indirizzo IP e da una porta. Le risorse sono specificate dal loro pathname nel filesystem. La reputazione di ogni servent è rappresentata da una terna di valori: il primo rappresenta l'identificativo del server, gli altri due valori rappresentano il numero di interazioni positive e negative, rispettivamente, effettuate. Analoga la sezione dedicata alla credibilità dei servent. I comandi accettati, ognuno dei quali occupa una riga, sono i seguenti:

- **sleep(n.sec)**: lascia il software in attesa degli eventi per il numero di secondi specificato.
- **search(criteri)**: effettua una ricerca nella rete Gnutella specificata dal criterio.
- **poll(**Sid_1 Sid_2 **...** Sid_n**)**: indice una votazione per gli identificativi Sid_1, Sid_2 ... Sid_n.
- **end**: termina il programma

La Figura 4.10 mostra un esempio di file di configurazione.

```
#esempio di configurazione
[ CONFIGURATION ]
listen_port=6348
web_port=12346
max_incoming_connections=2
max_outgoing_connections =1
lofile = "./reputella.log"

[ FILES ]
nomefile1
nomefile2

[ REPUTATION ]
8b6dc026ff887927d22c46e889a43ca5    1 3
03a4bf8cfa6fd356ba657c819277d5e9 12 1
595c12254c651fd8104a79f0b0151bb4 17 2

[ CREDIBILITY ]
8b6dc026ff887927d22c46e889a43ca5    5 3
03a4bf8cfa6fd356ba657c819277d5e9 2 7
55823ab04ea71c2f1cfaa55f37c99009 3 1
797fa867893fbcec49dd17fd4ee6f45f 42 2

[ CONNECTIONS ]

#159.149.70.90 6346
#159.149.70.90 6347
127.0.0.1 6346

# elenco di comandi
[ COMMANDS ]
sleep(10)
query(jan anderson)
sleep(40)
poll(8b6dc026ff887927d22c46e889a43ca5 797fa867893fbcec49dd17fd4ee6f45f)
sleep(30)
end
```

Figura 4.10: File di configurazione

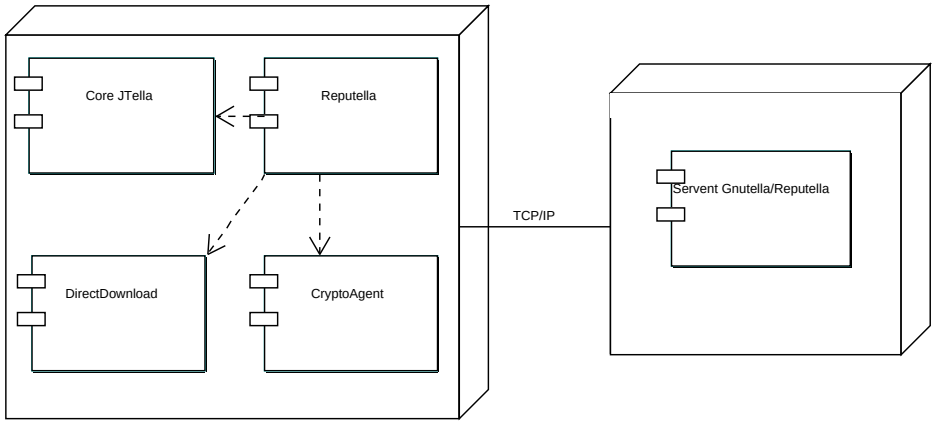


Figura 4.11: Diagramma delle componenti e messa in opera

4.1.7 Diagramma delle componenti e messa in opera

La Figura 4.11 unifica il diagramma di messa in opera con quello delle componenti:

Un servent Reputella è messo in opera su un solo computer ma interagisce, tramite rete TCP/IP, con altri servent Gnutella o Reputella. Il pacchetto è composto da tre parti: un substrato JTella, che garantisce la compatibilità all'indietro con il protocollo standard, un pacchetto crittografico (`CryptoAgent`) ed infine *Reputella*, il cuore dell'estensione, dipendente dagli altri due.

4.2 XP2PRep : la reputazione delle risorse

Come visto in precedenza è possibile attribuire la reputazione alle risorse modificando in parte il protocollo P2PRep. Il protocollo risultante, chiamato XP2PRep, prevede che per riferirsi alle risorse venga utilizzato il digest del loro contenuto. I nomi dei file, infatti, non sono identificativi univoci e universali del loro contenuto. La scelta, nell'implementazione prototipale di questa estensione, è caduta su MD5 [64].

Siccome le votazioni avvengono richiedendo informazioni sulle risorse e queste sono identificabili solo dal loro digest, è necessario che i messaggi `QueryHit` contengano, oltre al nome del file, anche questo ulteriore campo di 128 bit. La scelta di fondo di garantire l'interoperabilità dei protocolli ci ha spinto ad utilizzare il campo privato del trailer per elencare, come illustrato in Figura 4.12, i digest di tutti i file descritti.

Ad ogni file infatti, corrisponde un record nel *result set*, formato da indice, dimensione e nome di file. Nel campo trailer vengono elencati, per ognuno dei file descritti i digest MD5, codificati in 16 bytes ciascuno. In questo modo si garantisce che i messaggi `QueryHit`, possano essere compresi da chiunque, indipendentemente dal fatto che utilizzino o meno il protocollo esteso. Infatti, i server tradizionali potrebbero semplicemente ignorare il campo trailer e considerare le risposte estese come normali `QueryHit`.

I messaggi `XPoll` si differenziano dai messaggi `Poll` perché, invece di riportare il digest della chiave pubblica dei servent, elencano i digest delle risorse da valutare. I messaggi `XPoll` terminano con la chiave pubblica associata alla sessione. Le risposte `XPollReply` elencano i voti attribuiti ai file in un campo crittato con la chiave prelevata dal messaggio `XPoll`.

Si nota in questa descrizione come sia semplice modificare un client P2PRep per implementare

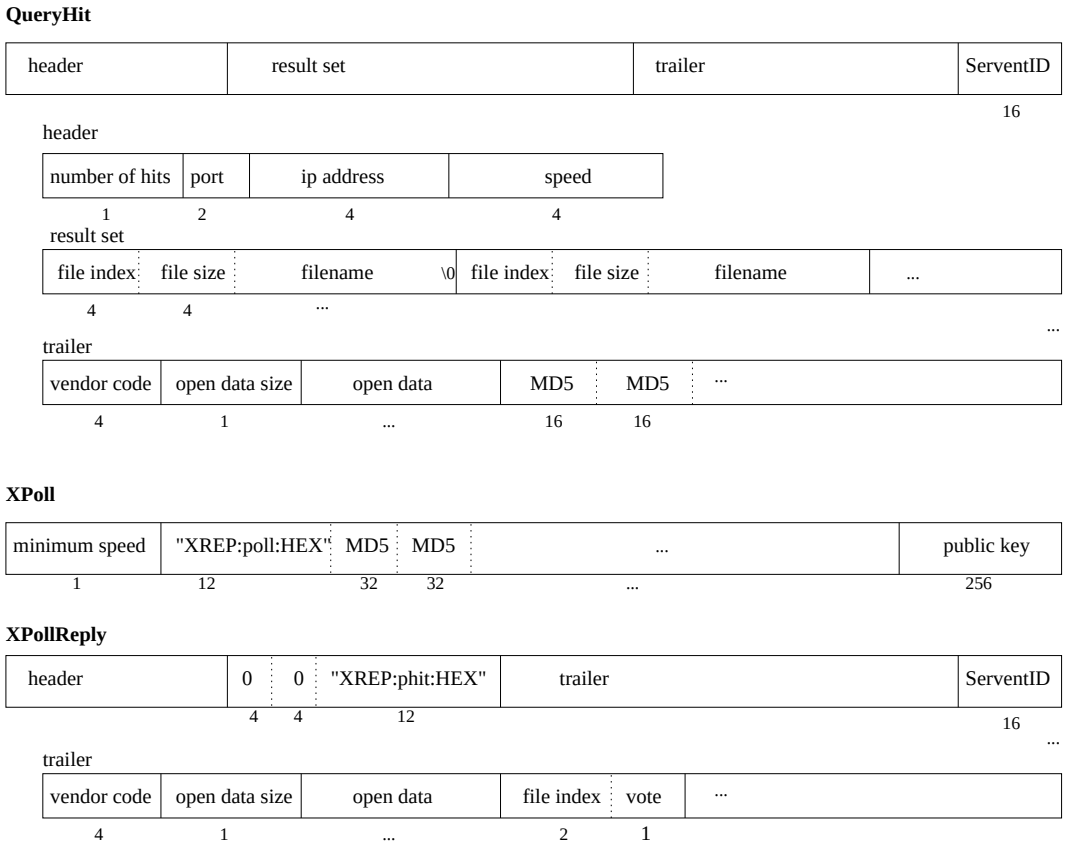


Figura 4.12: Schema dei messaggi di XP2PRep

il protocollo basato sulle risorse, poiché possono essere riutilizzate tutte le strutture di gestione dell'esperienza, della crittografa e gran parte della codifica dei messaggi.

4.3 Prototipi

Per l'implementazione dei prototipi è stata utilizzata la libreria *Reputella 0.1* [58], ottenuta modificando la libreria di API Java *Jtella 0.7* [45]. Jtella mette a disposizione tutte le funzioni necessarie per connettersi ad una rete Gnutella e per mandare messaggi Multicast e Unicast. Reputella eredita le classi di JTella, ed oltre ad implementare il protocollo Gnutella [73], implementa anche le estensioni *Dati Privati* e i messaggi necessari al funzionamento del protocollo P2PRep. Si riporta in seguito la descrizione di due prototipi, uno atto a dimostrare la semplicità con la quale si possa scrivere un software che monitorizzi il comportamento della rete Gnutella utilizzando la libreria Reputella, l'altro che implementi una sessione di ricerca completa di votazione, come specificato nel protocollo P2PRep.

4.3.1 Monitor: un'applicazione per Gnutella

Un esempio di programma funzionante che utilizzi le classi reputella è `Monitor`, i cui sorgenti sono visibili nell'appendice A. `Monitor` è un programma che si connette alla rete Gnutella e produce in output l'elenco di tutti i file offerti visibili in rete. Quando il programma riceve messaggi `QueryHit` produce su standard output un file CSV (Comma Separated Values) contenente le seguenti informazioni:

- IP del nodo d'origine del messaggio.
- Velocità dichiarata dal nodo.
- Nome e dimensione dei file offerti.

La classe principale è `ReplyMonitor`, che si occupa di generare la connessione verso un nodo specificato e definirne le caratteristiche. La connessione è gestita da un'istanza della classe `GNUTellaConnection`.

All connessione così definita viene associata,tramite metodo getSearchMonitorSession(), un'istanza della classe Receiver, dedicata alla gestione dei messaggi in arrivo. Questo oggetto, istanza della classe TestReceiver, implementa soltanto il metodo receiveSearchReply(), l'unico utile per i fini del programma.

4.3.2 Reputella, il prototipo del protocollo P2PRep

Il prototipo del protocollo P2PRep è articolato su tre file i cui sorgenti sono inclusi nell'appendice B.

Questo prototipo serve a dimostrare l'interoperabilità del protocollo P2PRep con la rete Gnutella e permette di inoltrare e ricevere, oltre ai tradizionali messaggi Gnutella, anche i messaggi `Poll` e `PollHit`. Compilando le tre parti si ottengono due programmi eseguibili: `PSearchExample` e `PReplyExample`. Entrambi questi oggetti fanno riferimento alla classe `XReceiver`, che implementa la componente `Receiver`.

I due programmi si connettono alla rete Gnutella usando il metodo addConnection() della classe `GRouter`, derivata dalla classe `GNUTellaConnection`.

PSearchExample si occupa di inoltrare un messaggio `Query` contente la stringa di ricerca: "stairway heaven txt". Per farlo usa il metodo `createSearchSession()` della classe `GRouter`. Poi si mette in attesa di ricevere risposte, che gestisce con un'oggetto `XReceiver`.

PReplyExample, complementare a PSearchExample, si occupa invece di rispondere ai messaggi `Query` con un messaggio `QueryHit`.

La gestione dei messaggi è completamente implementata nella classe `XReceiver`, comune ad entrambi i programmi. `XReceiver`, tramite i metodi `XreceiveSearch`, `XreceiveSearchReply`, `XreceivePoll` e `XreceivePollReply` è in grado di manipolare tutti i messaggi Unicast e Multicast definiti dal protocollo P2PRep. I messaggi `Ping` e `Pong` vengono gestiti automaticamente dalla libreria Reputella.

Quando PSearchExample riceve le risposte alla ricerca effettuata da parte delle istanze di PReplyExample, prepara ed inoltra un messaggio `Poll` contenente gli identificativi dei nodi offerenti. PReplyExample, al ricevimento di questi messaggi, risponde a sua volta con messaggi `PollHit`.

Questo prototipo implementa, secondo il protocollo P2PRep tutti i messaggi necessari, gestisce correttamente tutte le parti crittografiche e garantisce la persistenza delle chiavi. La persistenza e la gestione dei database dell'esperienza non sono visibili nell'attuale prototipo, pur esistendo, nella libreria Reputella, classi che le implementano.

Esiste anche un prototipo che implementa il protocollo XP2PRep in grado di dimostrare l'interoperabilità del protocollo XP2PRep con il protocollo Gnutella. Questo prototipo prevede inoltre

che le sue funzionalità siano controllabili da un file di configurazione, con il quale è possibile definirne il comportamento.

Grazie a questi prototipi si è dimostrata l'attuabilità del protocollo P2PRep, e si è fornita alla comunità un insieme di classi di pubblico dominio, utili per lo sviluppo di software Gnutella.

Capitolo 5

Conclusioni

Il mentitore dovrebbe tener presente che per essere creduto non bisogna dire che le menzogne necessarie.

Italo Svevo

È stata presentata in questa tesi un'analisi delle problematiche legate alle reti peer-to-peer in relazione alla necessità di regolare le interazioni sulla base delle reputazioni acquisite. Sono stati presentati diversi esempi che illustrano le soluzioni adottate in ambienti client/server per includere il concetto di reputazione. A questi è seguita un'analisi dei modelli esistenti in letteratura adatti ad architetture decentralizzate. Sono state analizzate le reti peer-to-peer, illustrando le tipologie di reti presenti. Lattenzione è stata rivolta poi in particolare a Gnutella, sulla quale il lavoro di tesi è basato, e della quale sono stati descritti l'architettura, i messaggi necessari al suo funzionamento e le sue debolezze. È stato quindi proposto un protocollo che estende Gnutella, includendo la possibilità di esprimere dei voti e di ricevere da parte della comunità dei giudizi sui servent con i quali si potrebbero effettuare le transazioni. Il protocollo poposto, chiamato P2PRep, è semplice da implementare e in grado di integrarsi in modo trasparente nella rete Gnutella esistente. Di questo protocollo sono state analizzate le caratteristiche, in particolare in relazione agli attacchi più comuni perpetrati ai danni degli ambienti peer-to-peer. Sono state quindi descritte le principali caratteristiche dell'implementazione del protocollo, illustrando mediante diagrammi e schemi UML l'architettura tipica di un client Gnutella assieme alle modifiche necessarie per includere il protocollo P2PRep.

Per concludere aggiungiamo alcune considerazioni addizionali sul protocollo proposto che meritano di essere prese in considerazione.

- *Costi limitati* : L'implementazione non richiede grosse modifiche a client esistenti e l'appesantimento al protocollo è contentuto. Per ogni ricerca infatti, al più viene generata un'altra sessione broadcast di `Poll`, di dimensioni paragonabili ad una normale ricerca. Va considerato che, una volta trovato un insieme di servent ragionevolmente stabile, fidato e ricco, il numero di volte durante le quali ci sia l'esigenza di indire una votazione verrebbe ridotto in modo considerevole. Inoltre si ipotizza che l'aumento di banda dovuto alla presenza dei nuovi messaggi venga compensato dal minor numero di tentativi necessari per scaricare

una determinata risorsa. Per quanto riguarda invece lo spazio occupato su disco, questo è essenzialmente proporzionale al numero di servent di cui tenere traccia e come è stato misurato, resta ben al di sotto delle quantità di spazio occupate dalla media dei file presenti negli archivi condivisi.

- *Concentrazione del carico su pochi server* : Alcuni studi hanno dimostrato che la distribuzione delle risorse non è affatto uniforme, infatti esiste una porzione considerevole di nodi che si limita a prelevare risorse senza condividiere nulla (Free Riders), mentre una piccola parte di nodi detiene la maggior parte delle risorse disponibili in rete. Il protocollo P2PRep potrebbe aumentare ulteriormente il carico di questi punti nevralgici della rete. Una possibile soluzione a questo problema potrebbe essere di lasciare che i servent che godono alta reputazione, invece di farsi carico del trasferimento dei file, si possano limitare a fornire i loro digest. Il download può avvenire quindi con qualunque nodo, anche privo di fiducia. La qualità del file può essere infatti verificata al termine dello scaricamento, semplicemente verificandone il digest.

- *Free riding*: Un fenomeno che spesso si cerca di contrastare in ambienti tipo Gnutella è denominato "free riding" e si riferisce al problema di riscontrare in rete una parte considerevole di nodi che non esportano quasi alcuna risorsa e sfruttano la rete senza offrire nulla in cambio. Esistono diverse tecniche e proposte per diminuire l'impatto del problema. In certi casi si esige un numero minimo di file condivisi per poter accedere a determinate risorse, oppure vengono impedite le connessioni a chi non offra sufficienti file. Il problema tuttavia non ha una soluzione univoca, semplice e generale. Il protocollo P2PRep non soffre dell'esistenza di questi Free Riders, perché permette loro di essere comunque utili. Anche senza offrire risorse, dispendiose in termini di banda, ogni nodo può infatti condividere la propria esperienza segnalando, tramite votazioni, i servent di buona qualità con i quali sono state portate a termine transazioni. È interessante notare come i due giudizi (reputazione e credibilità) presenti nel protocollo P2PRep riflettano i comportamenti rispettivamente *server oriented* e *client oriented*, risaltando, in entrambi i casi, la correttezza del comportamento.

Il lavoro di tesi rappresenta solo un primo passo verso la realizzazione di un sistema flessibile e sicuro per la gestione della reputazione in ambienti peer-to-peer e lascia spazio per future estensioni.

Molto lavoro deve ancora essere fatto, soprattutto per utilizzare alcune delle nuove estensioni proposte dal protocollo. Un breve elenco delle possibili direzione nelle quali il lavoro di tesi potrebbe seere esteso include:

- *Gestione XML delle metainformazioni.* Queste informazioni permettono di descrivere il contenuto dei file e quindi di effettuare ricerche non solo relative ai nomi dei file, ma anche al loro contenuto. Sarebbe possibile ottenere, ad esempio, l'elenco di tutte le musiche composte in un particolare anno oppure di tutti i documenti che trattano di sicurezza informatica, pur non avendo particolari parole chiave nel nome del file. Per ogni tipo di file può essere definito un particolare documento XSD, schema di documenti XML, in grado di trattare diversi tipi di dati. LimeWire, azienda che ha implementato nel suo client le metainformazioni, mette a disposizione alcuni schemi XSD per file audio e video. Si potrebbe quindi definire uno schema XSD in grado di gestire la reputazione, sia rispetto ai servent, sia rispetto alle

risorse. [1] In generale i messaggi `Poll` potrebbero essere implementati semplicemente da ricerche con particolari metainformazioni riguardanti la reputazione.

- *Token per la reputazione.* Un altro modo per sfruttare l'esperienza nelle reti peer-to-peer è quella di fornire ad ogni servent, dopo ogni transazione ben riuscita, un token che dichiari l'avvenuta transazione. Questo metodo trae l'ispirazione dal protocollo Web of Trust di PGP. I protocolli P2PRep e XP2PRep possono essere estesi per comprendere questa estensione, applicando loro le seguenti modifiche.

  Qualora un servent veda il proprio `ServentID` comparire in un messaggio `Poll` risponde al mittente con un messaggio di tipo `TokenHit` contente i token ricevuti. Il Token è composto da una data, un giudizio, la firma digitale e il `ServentID` di chi lo genera. Qualora il ricevente del messaggio abbia nel proprio database della credibilità il `ServentID` specificato e la sua corrispondente chiave pubblica (andrebbe aggiunta una tabella che associ ad ogni `ServentID` contattato la sua chiave pubblica), potrebbe facilmente verificare l'integrità del giudizio ed eventualmente raffrontarla al valore di credibilità del nodo che ha espresso il giudizio.

  Ovviamente questi token possono rappresentare solo giudizi positivi, dal momento che eventuali giudizi negativi verrebbero eliminati. Un altro possibile metodo per usare questi token prevede il loro inserimento nei messaggi di ricerca delle risorse. Si potrebbe fare in modo che ogni servent preveda delle connessioni privilegiate esclusivamente per coloro che possono dimostrare di avere una reputazione significativa.

  La data associata al token serve a far decadere il valore di quelli troppo vecchi che, in ambienti come Gnutella, non hanno molto valore. Oppure potrebbe essere utile poter dimostrare una lunga tradizione di buoni comportamenti.

  Un problema legato a questa soluzione è che risulterebbero pubblici dei giudizi che implicano l'avvenuta transazione tra diversi servent, perdendo parte dell'anonimato. Inoltre, perchè abbia senso, devono essere spediti decine di token, ognuno di circa 150 byte.

- *Supernodi.*

  Un'altra possibile estensione riguarda la possibilità di includere nel protocollo P2PRep la gestione dei Supernodi. L'interesse da parte della comunità scientifica ed industriale verso le topologie ibride (Cfr. sez. 2.1.2) è infatti crescente. FastTrack [28], grazie al suo modello a Supernodi, ha mostrato le grandi potenzialità di questa struttura, in grado di fornire grandi potenzialità di scalabilità. Anche Gnutella, con il protocollo Ultrapeer [6], si allinea a questa tendenza, includendo in modo trasparente nella rete la presenza di Supernodi. Questa soluzione, ottima per migliorare la scalabilità complessiva della rete e in grado di migliorarne la velocità media, presenta però nuovi problemi legati alla reputazione. La scelta di diventare un Supernodo è demandata alla volontà di ogni client, il quale, in condizioni favorevoli di banda e di memoria, assume il nuovo ruolo, indipendentemente dalla sua reputazione. I nodi che vogliano comportarsi da foglie non hanno, d'altra parte, alcun metodo per discriminare eventuali Supernodi disonesti. Siccome la connessione di una foglia verso la rete Gnutella è completamente determinata dalla volontà del Supernodo al quale è connessa, si comprende l'importanza di estendere il protocollo P2PRep anche a questo ambito.

[1] Le varie proposte XSD di LimeWire prevedono già, per ogni file, la possibilità di definirne un digest (viene usata la funzione SHA1), che premetterebbe di ridefinire il protocollo XP2PRep (basato sulle risorse) in modo piuttosto semplice: ogni risposta deve contenere il digest delle risorse.

D’altra parte, una volta stabilito un certo livello di fiducia nei confronti di un Supernodo, oltre alle tabelle di routing si potrebbe demandare anche la gestione della reputazione dei vari nodi della rete.

Bibliografia

[1] Alfarez Abdul-Rahman. The pgp trust model. In *EDI-Forum: The Journal of Electronic Commerce*, April 1997.

[2] Alfarez Abdul-Rahman and Stephen Hailes. A distributed trust model. pages 48–60.

[3] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *HICSS*, 2000.

[4] Advogato trust metric. http://www.advogato.org/trust-metric.html.

[5] Amazon.com. http://www.amazon.com.

[6] Christopher Rohrs Anurag Singla. Ultrapeers: Another step towards gnutella scalability,working draft. http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm.

[7] Abel A. Arthur. Optimizing routing in distributed networks. In *Proceedings of the Seventh International Conference on Information and Knowledge Management (CIKM 1999)*, 1999.

[8] Audiogalaxy. http://www.audiogalaxy.com.

[9] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: A survey, 1995.

[10] Bearshare. http://www.bearshare.com.

[11] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945: Hypertext Transfer Protocol — HTTP/1.0, May 1996. Status: INFORMATIONAL.

[12] Rainer Burkhardt. *UML: Unified Modeling Language*. Addison-Wesley, 1997.

[13] B. N. Levine C. Shields. A protocol for anonymous communication over the internet. In *Proc. 7th ACM Conference on Computer and Communication Security(ACM CCS 2000)*, November 2000.

[14] M. Carmack. P2P and the W3C. http://www.w3.org/2001/04/w3c-p2p.

[15] with ideas from Vincent Falco Christopher Rohrs. The ping / pong scheme. http://rfc-gnutella.sourceforge.net/Proposals/Ping-Pongcheme.htm.

[16] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2000.

[17] Dati sull'internet nel mondo. http://www.gandalf.it/dati/dati1.htm.

[18] Debian. http://www.debian.org/.

[19] L. Deutsch. Zlib compressed data format specification version 3, 1950.

[20] D.Gambetta. Can we trust? In *Trust: Making and Breaking Cooperative Relations*, Oxford, 1990.

[21] R. Dingledine, M.J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, California, USA, July 2000.

[22] Directconnect. http://www.neo-modus.com.

[23] ebay. http://www.ebay.com.

[24] What you need to know about echelon. http://news.bbc.co.uk/hi/english/sci/tech/ /newsid_1357000/1357513.stm.

[25] Edonkey 2000. http://www.edonkey2000.com.

[26] Little endian, big endian. OID.2.6.53.

[27] Epinions. http://www.epinions.com.

[28] Fasttrack. http://www.fasttrack.nu.

[29] Gnutella servers and host cache. http://www.gnutella.co.uk/servers.

[30] Gnutelliums. http://www.gnutelliums.com.

[31] L. Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, May/June 2001.

[32] Google. http://www.google.org.

[33] Gnu gnerale public license. http://www.gnu.org/licenses/gpl.txt.

[34] Christopher Rohrs Greg Bildson. An extensible handshaking protocol for the gnutella network. Technical report, Lime Wire LLC, 2001. http://rfc-gnutella.sourceforge.net/Proposals/Handshake_06/Gnutella06.txt.

[35] Paul Harrison. The circle: Some first steps towards decentralizing internet services. In *Linux Conference of Australia*, 2002. http://yoyo.cc.monash.edu.au/ pfh/circle/.

[36] P. Hoffman. RFC 2487: SMTP service extension for secure SMTP over TLS, January 1999. Status: PROPOSED STANDARD.

[37] D. Reed J. Oikarinen. RFC 1459: Internet Relay Chat Protocol, May 1993. Status: INFORMATIONAL.

[38] Zoran Despotovic Karl Aberer. Managing trust in a peer-2-peer information system. In *Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM 2001)*, 2001.

[39] Kazaa. http://www.kazaa.com.

[40] Limewire. http://www.limewire.org.

[41] A. D. Rubin M. K. Reiter. Crowds: Anonymity for web transactions. In *ACM Transactioins on Information and System Security*, November 1998.

[42] D. Goldschlag M. Reed, P. Syverson. Proxies for anonymous routing. In *12th Annual Computer Security Applications Conferences*, December 1995.

[43] Raphael Manfredi. Introducing new bye (0x02) packet in gnutella 0.4. http://rfc-gnutella.sourceforge.net/Proposals/BYE/Bye.txt.

[44] Aviel D. Rubin Marc Waldman and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, August 2000.

[45] Ken McCrary. Jtella: Java api for the gnutella network. http://www.kenmccrary.com/jtella.

[46] Inc. Michael T. Prinkey, Aeolus Research. An efficient scheme for query processing on peer-to-peer networks. http://aeolusres.homestead.com/files/index.html.

[47] Nelson Minar. Distributed systems topologies: Part 1. http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html.

[48] G. Mohr. Hash/urn gnutella extensions (huge) v0.93. http://rfc-gnutella.sourceforge.net/Proposals/HUGE/draft-gdf-huge-0_93.txt.

[49] Mpeg audio layer-3. ISO/IEC 11172-3:1993 Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio.

[50] Napster. http://www.napster.com.

[51] Status Of. Deflate compressed data format specification version 1.3.

[52] Opennapster. http://www.opennapster.org.

[53] Peer-to-peer working group. http://www.peer-to-peerwg.org/whatis/.

[54] Poblano - a distributed trust model for peer-to-peer networks. http://www.jxta.org/project/www/docs/trust.pdf.

[55] J. Postel. RFC 821: Simple mail transfer protocol, August 1982. See also STD0010. Obsoletes RFC0788. Status: STANDARD.

[56] F. PUB, National, and o Standards. Fips pub 180-1: Secure hash standard, 1995.

[57] Raphael. Tunneling in gnutella. http://rfc-gnutella.sourceforge.net/Proposals/ /TUNNELING/tunneling_discusion.txt.

[58] Reputella. http://seclab.dti.unimi.it/p2prep.

[59] Rfc 2810: Internet relay chat: Architecture. ftp://ftp.irc.org/irc/docs/rfc2810.txt.

[60] Rfc gnutella: an approach to define a standard. http://rfc-gnutella.sourceforge.net.

[61] Simple mail transfer protocol. http://www.faqs.org/rfcs/rfc2821.html.

[62] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. Technical Report TR-2001-26, University of Chicago, Department of Computer Science, July 2001.

[63] J. Ritter. Why gnutella can't scale. no, really., February 2001.

[64] R. Rivest. RFC 1321: The MD5 message-digest algorithm, April 1992. Status: INFORMATIONAL.

[65] Christopher Rohrs. Query routing for the gnutella network. http://rfc-gnutella.sourceforge.net/Proposals/QRP/query_routing.htm.

[66] Christopher Rohrs. Some messages about guid tagging. http://rfc-gnutella.sourceforge.net/Development/GUID-Tagging.htm.

[67] Slashdot, news for nerds. stuff that matters. http://slashdot.org.

[68] W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major seti project based on project serendip data and 100,000 personal computers. In *Astronomical and Biochemical Origins and the Search for Life in the Universe, Fifth Intl. Conf. on Bioastronomy*, Editrice Compositori, Bologna, Italy, 1997.

[69] P F Syverson, D M Goldschlag, and M G Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, 4–7 1997.

[70] A.S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall International,Inc, 1992.

[71] Sumeet Thadani. Meta information searches on the gnutella network. http://rfc-gnutella.sourceforge.net/Proposals/MetaData/meta_information_searches.htm.

[72] Sumeet Thadani. Meta information searches on the gnutella network (addendum). http://rfc-gnutella.sourceforge.net/Proposals/MetaData/meta_information_searches

[73] *The Gnutella Protocol Specification v0.4 (Document Revision 1.2)*, June 2001. http://www.clip2.com/GnutellaProtocol04.pdf.

[74] Jason Thomas. Gnutella generic extension protocol (ggep), version 0.31. http://rfc-gnutella.sourceforge.net/Proposals/GGEP/GnutellaGenericExtensionProtocol.0.31.htm.

[75] Clay Shilds Vincent R. Scarlata, Brian N. Levine. Responder anonymity and anonymous peer-to-peer file sharing. Technical report, 2000.

[76] M. Waldman and D. Mazières. Tangler: A censorship-resistant publishing system based on document entanglements. In *Proc. of the 8th ACM Conference on Computer and Communications Security*, Philadelphia, Pennsylvania, USA, November 2001.

[77] Winmx. www.musiccity.com.

[78] Wired news. http://www.wired.com/news.

[79] B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *Proc. of the 27th International Conference on Very Large Data Bases*, Rome, Italy, September 2001.

[80] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.

Appendice A

Sorgenti del Monitor

Monitor è un programma in grado di registrare le informazioni relative a tutti i messaggi `QueryHit` che vengono inoltrati dal nodo durante una sessione Gnutella. Il software, scritto interamente in Java, usa la libreria Reputella [58].

```
/*
 * Reputella Monitor
 *
 * Copyright (c) 2001/2002 Fabrizio Cornelli, All Rights Reserved.
 * Copyright (c) 2000 Ken McCrary, All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON-COMMERCIAL purposes and without
 * fee is hereby granted provided that this copyright notice
 * appears in all copies.
 *
 */

import com.P2PRep.reputella.*;


/**
 *  An example class for showing how to monitor search queries on the
 *  GNUTella network using reputella. The application expects two command
 *  ine parameters, the host name, and the port the servant is listening on
 *
 */
public class ReplyMonitor {
    private GNUTellaConnection conn;

    /**
     *  Constructs the example given a started network connection
     *
     */
    public ReplyMonitor(GNUTellaConnection networkConnection) {
        this.conn = networkConnection;
    }
```

```
/**
 *  Main entrypoint for the example
 *
 */
public static void main(String[] args) {
    System.out.println("<--- RepuTella MonitorExample running --->\n");

    if (args.length != 2) {
        System.out.println("Usage: Monitor host port");
        System.exit(1);
    }

    try {
        System.out.println("Connecting to Gnutella Network...");
        System.setProperty("RepuTella.logQueryHits","true");
        //-------------------------------------------------------------
        // Start a network connection and listen for succesful connection
        //-------------------------------------------------------------
        GNUTellaConnection c;
        ConnectionData cd=new ConnectionData();
        cd.setOutgoingConnectionCount(8);
        cd.setIncommingConnectionCount(8);

        /* the main class: it collects all the connections */
        c = new GNUTellaConnection(cd,args[0],
        Integer.decode(args[1]).intValue());


        c.getSearchMonitorSession(new TestReceiver());
        c.start();
    }
    catch (Exception e) {
        e.printStackTrace();
    }

}

/**
 *  Test class for monitoring query messages, prints out queries to the console
 *
 */
static class TestReceiver extends MessageReceiverAdapter {
    /**
     *  Receives SearchReplies messages from the network
     *
     *  @param searchMessage a search message received on the network
     */
    public void receiveSearchReply(SearchReplyMessage s) {
        System.out.println("SearchReply Session received: " + s);
        for(int i=0;i<s.getFileCount();i++){
```

```
                System.out.print(s.getIPAddress()+","
+s.getDownloadSpeed()+","+s.getPort()+",");
                System.out.println(s.getFileRecord(i).getName()
+","+s.getFileRecord(i).getIndex()+","
+s.getFileRecord(i).getSize());
            }
        }

        /*public void receiveSearch(SearchMessage s) {
            System.out.println("Search Session received: " +
            s.getSearchCriteria());

        }*/

    }
}
```

Appendice B

Sorgenti del Monitor

Monitor è un programma in grado di registrare le informazioni relative a tutti i messaggi `QueryHit` che vengono inoltrati dal nodo durante una sessione Gnutella. Il software, scritto interamente in Java, usa la libreria Reputella [58].

```
/*
 * Reputella Monitor
 *
 * Copyright (c) 2001/2002 Fabrizio Cornelli, All Rights Reserved.
 * Copyright (c) 2000 Ken McCrary, All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this software
 * and its documentation for NON-COMMERCIAL purposes and without
 * fee is hereby granted provided that this copyright notice
 * appears in all copies.
 *
 */

import com.P2PRep.reputella.*;


/**
 *  An example class for showing how to monitor search queries on the
 *  GNUTella network using reputella. The application expects two command
 *  ine parameters, the host name, and the port the servant is listening on
 *
 */
public class ReplyMonitor {
    private GNUTellaConnection conn;

    /**
     *  Constructs the example given a started network connection
     *
     */
    public ReplyMonitor(GNUTellaConnection networkConnection) {
        this.conn = networkConnection;
    }
```

```
/**
 *  Main entrypoint for the example
 *
 */
public static void main(String[] args) {
    System.out.println("<--- RepuTella MonitorExample running --->\n");

    if (args.length != 2) {
        System.out.println("Usage: Monitor host port");
        System.exit(1);
    }

    try {
        System.out.println("Connecting to Gnutella Network...");
        System.setProperty("RepuTella.logQueryHits","true");
        //-------------------------------------------------------------
        // Start a network connection and listen for succesful connection
        //-------------------------------------------------------------
        GNUTellaConnection c;
        ConnectionData cd=new ConnectionData();
        cd.setOutgoingConnectionCount(8);
        cd.setIncommingConnectionCount(8);

        /* the main class: it collects all the connections */
        c = new GNUTellaConnection(cd,args[0],
        Integer.decode(args[1]).intValue());


        c.getSearchMonitorSession(new TestReceiver());
        c.start();
    }
    catch (Exception e) {
        e.printStackTrace();
    }

}

/**
 *  Test class for monitoring query messages, prints out queries to the console
 *
 */
static class TestReceiver extends MessageReceiverAdapter {
    /**
     *  Receives SearchReplies messages from the network
     *
     *  @param searchMessage a search message received on the network
     */
    public void receiveSearchReply(SearchReplyMessage s) {
        System.out.println("SearchReply Session received: " + s);
        for(int i=0;i<s.getFileCount();i++){
```

```
                System.out.print(s.getIPAddress()+","
+s.getDownloadSpeed()+","+s.getPort()+",");
                System.out.println(s.getFileRecord(i).getName()
+","+s.getFileRecord(i).getIndex()+","
+s.getFileRecord(i).getSize());
            }
        }

        /*public void receiveSearch(SearchMessage s) {
            System.out.println("Search Session received: " +
            s.getSearchCriteria());

        }*/

    }
}
```