Parole chiave:

P2P

Peer Data Management Systems

Gestione ad anello

Gestione a ipercubo

RINGRAZIAMENTI

Desidero ringraziare la Professoressa Sonia Bergamaschi per il prezioso aiuto fornitomi durante lo svolgimento della tesi, il dott. ing. Francesco Guerra e lo studente Matteo Magni per la loro disponibilità.

Un sincero ringraziamento va alla mia famiglia a tutti i miei familiari per il loro costante sostegno durante il percorso di studi.

Non posso non ringraziare tutti i miei compagni di corso e di studio: Mattia e Luca, che mi sopportano dai tempi delle superiori, Bompa, Elena, Enri, Fabiana, From, Matteo, Mauri, Musso, Tania con i quali ho condiviso gioie e fatiche in questi anni di studio.

Desidero ringraziare tutti i miei amici, ma, in particolare, Andrea, Giulia e Stefano, che mi hanno sostenuto e aiutato durante i miei momenti di crisi, e tutte le persone che il Signore mi ha donato e che mi hanno accompagnato in questi anni intensi.

A tutti va un enorme e sentito…GRAZIE…

Matteo Caruso

INDICE

INDICE DELLE FIGURE

INTRODUZIONE

Una rete Peer-to-Peer (P2P) permette ad un gruppo di elementi distribuiti, chiamati peer, di connettersi fra loro e di condividere le risorse in loro possesso. All'interno della rete, ciascun peer ha capacità e responsabilità equivalenti. Ciò differisce dalle architetture di tipo client/server, in cui alcuni computer sono dedicati al servizio di altri computer.

Le reti P2P presentano alcune peculiarità che generano profonde differenze rispetto alle altre reti, come, ad esempio, il Web: non prevedono l'utilizzo di un browser Web per poter navigare attraverso la rete stessa; per poter comunicare con gli altri nodi della rete, è necessario utilizzare un software, che implementi un protocollo di comunicazione (Gnutella, FastTrack,…) fra i peer che costituiscono la rete. Ciascun nodo funge sia da client che da server, a differenza dell'architettura Web, dove solo alcuni nodi sono specificatamente volti al soddisfacimento della richieste di altri nodi.

Le reti P2P prevedono alcuni vantaggi:

1. si basano sui protocolli http, ftp,…, i quali prevedono un libero scambio di informazioni attraverso i router e i firewall
2. non richiedono operazioni di autenticazione; fondamentale per garantire l'anonimità degli utenti della rete
3. danno la possibilità di implementare chat (ormai sempre più di moda) e di utilizzare le e-mail
4. non necessitano di amministratori di rete e sono particolarmente semplici da utilizzare. Inoltre, il costo di aggiornamento delle informazioni, che circolano nella rete, è praticamente nullo
5. sono estremamente tolleranti ai guasti, perché non dipendono da una struttura centrale, quindi la perdita di un nodo non comporta un isolamento totale dalle informazioni, a cui si desidera accedere

Non mancano, però, gli svantaggi:

1. è necessario realizzare e implementare continuamente nuovi ed efficienti algoritmi di ricerca
2. la larghezza di banda impone delle restrizioni sul possibile numero di utenti che possono accedere alla rete. Ciascun nodo ha la libertà di stabilire quanti nodi possono collegarsi ad esso: un numero troppo elevato di collegamenti porta ad un consumo eccessivo della sua larghezza di banda, mentre un numero esiguo di utenti porta ad una vera e propria restrizione sulla possibilità di accesso alla rete da parte di nuovi nodi
3. non si hanno garanzie sulla qualità e sull'affidabilità dei dati, che vengono forniti dal nodo. Alcune risorse possono essere fittizie, aumentando, così, il rischio della diffusione dei virus

Gli aspetti di ricerca interessanti nell'ambito delle Basi di Dati sono relativi alla gestione e alla ricerca di dati in reti P2P. Tali dati devono essere gestiti attraverso appositi sistemi, definiti Peer Data Management Systems (PDMS), che permettono la condivisione di dati eterogenei in modo distribuito e scalabile. Normalmente, per il numero di sorgenti trattate, tale condivisione si basa sulla definizione di mapping semantici tra gli elementi delle sorgenti coinvolte, piuttosto che sulla realizzazione di uno schema globale sintesi dei dati delle sorgenti coinvolte.
Lo scopo iniziale di questa tesi è quello di studiare e confrontare fra loro le principali applicazioni sviluppate su reti P2P a disposizione degli utenti. Successivamente, saranno presentate due interessanti strutture, una ad anello (Chord) e una a ipercubo (HyperCuP), che consentono di gestire i peer per quanto riguarda la comunicazione, la condivisione e lo scambio dei dati.

Il testo è composto dai seguenti 3 capitoli:

1) Panoramica delle applicazioni peer to peer (P2P) più utilizzate

In questo capitolo vengono presentate le più diffuse reti P2P: Gnutella1, Gnutella2, FreeNet, Napster e, in particolare, KaZaA, la quale è stata studiata con un efficace e intenso studio di misurazione.

2) Ricerca di file in architettura P2P

In questo capitolo viene inizialmente presentato il sistema Chord, studiando i meccanismi di ricerca della risorsa, di ingresso e di uscita dalla rete e mostrando come installare una versione del sistema sul Sistema Operativo del nostro computer.

3) Architettura P2P basata sull'ontologia

In questo capitolo, viene presentato HyperCuP, dando particolare attenzione alla disposizione a ipercubo dei peer, alle sue proprietà, alla costruzione e al mantenimento della rete, allo sviluppo della topologia attraverso un routing basato sull'ontologia e presentando una implementazione del sistema, HyperCuP. In questa tesi vengono presentati anche la libreria digitale Elvis e il progetto FOAF-Realm. La prima, per trovare la risorsa cercata, fornisce all'utente tre tipi di ricerca: semplice, a catalogo e, in particolare, semantica, e utilizza la rete HyperCuP per la comunicazione L2L fra tutte le librerie digitali. Il secondo fornisce un'idea interessante su come gestire l'accesso alle risorse degli utenti: solo gli amici più stretti possono accedere alle informazioni.

CAPITOLO 1

PANORAMICA DELLE APPLICAZIONI PEER TO PEER (P2P) PIU' UTILIZZATE

1.1 Panoramica delle topologie P2P

1.1.1 Topologie di base

Tradizionalmente esistono diverse topologie di rete, in particolare, come illustrato in Figura 1, le topologie di base adottate nelle reti peer-to-peer possono essere: *centralizzate*, ad *anello*, *gerarchiche* e *decentralizzate*.

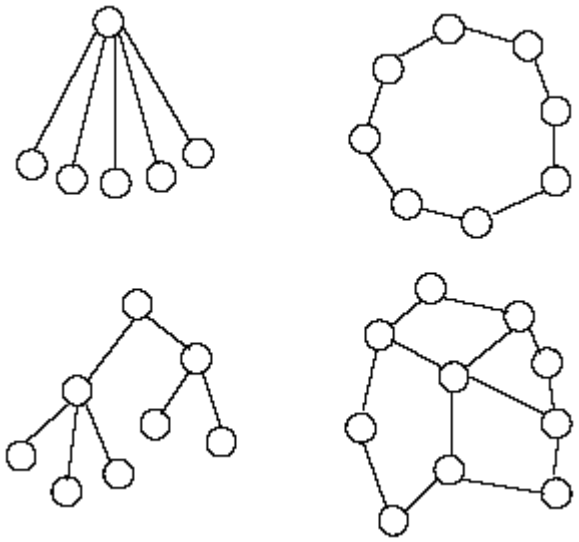


Figura 1: Topologie di base di reti peer to peer

- **Centralizzata:** in Figura 1a 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, estrae 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 singolo point of failure.

- **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 1b, 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.

- **Gerarchica:** i sistemi gerarchici, illustrati in Figura 1c, 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 servant. 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 da grafo non diretto nel quale tutti i nodi hanno un numero di archi variabile. Nella Figura 1d si nota l'assenza del server che aveva caratterizzato la Figura 1a, mentre risalta la presenza di nodi dello stesso tipo e grado.

  Reti di questo tipo devono poter risolvere diversi problemi; il primo dei quali è appunto il problema della prima connessione: in sistemi decentralizzati puri non dovrebbe esistere nemmeno un punto fisso di entrata, caratteristica raramente rispettata nelle implementazioni. Un altro problema fondamentale riguarda il routing dei messaggi, che richiede la specifica 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 servant di valutare la reputazione di altri servant 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.

1.1.2 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*.

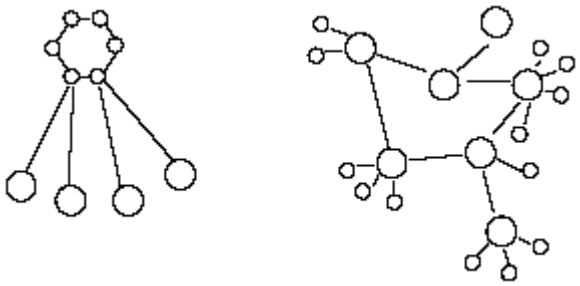


Figura 2: Topologie ibride di reti peer to peer

- **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 2a 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 si può far riferimento ai sistemi centralizzati.

- **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 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 pure e centralizzate, come Napster, tendono ad integrare nei protocolli di base delle funzioni ibride per tentare almeno di arginare alcuni dei problemi citati in precedenza. Naturalmente non essendo facile stabilire

quali servant possano 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 2b rappresenta il tipo di rete proposta da Limewire [5] per risolvere alcuni dei problema presenti in reti Gnutella: 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 che occorre studiare per poter capire quali siano le opportunità messe a disposizione dalle reti peer-to-peer. Vengono illustrati Gnutella1 e Gnutella2 (ambienti peer-to-peer puro), FreeNet (rete peer-to-peer anonima), Napster (rete peer-to-peer centralizzata) e KaZaA (implementazione commerciale molto efficiente e famosa).

1.2 La rete Gnutella1

Gnutella1 [3] è una rete peer-to-peer decentralizzata. Essa è attualmente utilizzata da milioni di utenti che, scaricando uno dei tanti programmi client che fanno uso di tale rete, possono cercare, ottenere e condividere vari tipi di file, ad esempio, mp3 e video.

E’stata sviluppata inizialmente come piccolo programma da Justin Frankel della

Nullsoft, una compagnia facente capo ad AOL più famosa per il suo player multimediale WinAMP. AOL terminò la sua attività nei primi mesi del 2000 ed ordinò alla Nullsoft di cessare lo sviluppo del software perché non si riusciva a controllare come gli utenti usavano Gnutella. Tuttavia i dettagli sul protocollo di Gnutella furono resi pubblici anche usando tecniche di reverse engineering e presto furono sviluppati nuovi client per accedere alla rete.
Al momento il protocollo Gnutella è sviluppato da un gruppo di programmatori chiamato 'Gnutella Developer Forum' , GDF (http://www.gnutellaforum.com). Il protocollo è OpenSource e così sono pure la maggior parte dei client.
Le caratteristiche salienti della rete Gnutella1 sono :

1) assoluta decentralizzazione (non è presente alcun server e tutti i nodi sono uguali)
2) rete autogestita
3) architettura aperta, protocollo semplice e flessibile

La maggior parte delle comunicazioni Internet avvengono su una base client-server. Il client si connette ad un server, che solitamente è più grande e veloce e dal quale reperisce informazioni (come ad esempio file).
Il protocollo di Gnutella è sostanzialmente diverso, poiché in tale protocollo i client diventano server ed i server diventano client allo stesso tempo. Il campo di gioco viene livellato e tutti possono essere client o server allo stesso tempo, non importa quanto grande o veloce siano. Per questo motivo, di solito ci si riferisce alla singola entità operante su rete Gnutella col nome di servant.
Tutto ciò si realizza creando una sorta di ambiente distribuito: ci si comporterà da server con le persone che desiderano scaricare file dal proprio computer, e ci si comporterà da client per scaricare i file da altri utenti. La rete Gnutella1 è fatta da migliaia (o milioni) di servant che comunicano tra di loro e inviano file attraverso la rete.
Ogni parte di informazione è chiamato pacchetto. Tutte le comunicazioni avvengono

con protocollo TCP/IP, non si fa uso del protocollo UDP.
I motivi del non utilizzo del protocollo UDP sono:

1) Il servizio di consegna dei pacchetti non è garantito perché alcuni possono essere persi o duplicati
2) Il servizio avviene senza connessione in quanto non vi è la fase di handshaking tra mittente e destinatario

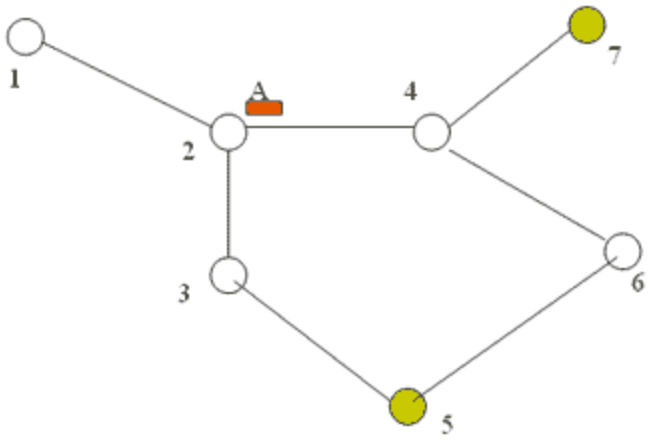
I motivi, invece, dell'utilizzo del protocollo TCP/IP sono:

1) Il servizio è orientato alla connessione: comprende l'instaurazione, l'utilizzo e la chiusura della connessione utilizzando la fase di handshaking
2) Il servizio è orientato al flusso di dati: considera il flusso dei dati dall'host mittente fino al destinatario
3) Il trasferimento dei dati avviene tramite un buffer: i dati sono memorizzati in un buffer e poi inseriti in un pacchetto quando il buffer è pieno
2) La connessione è full-duplex (bi-direzionale): una volta instaurata la connessione, è possibile il trasferimento contemporaneo in entrambe le direzioni

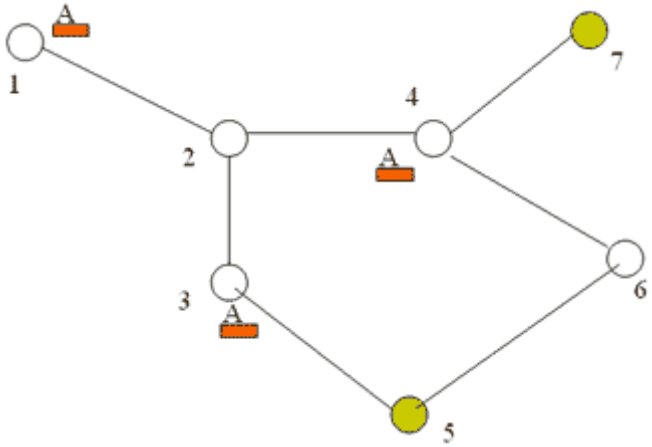
Per connettersi alla rete, bisogna conoscere due cose: l'indirizzo IP e la porta di uno qualsiasi dei servant che sono già connessi. Quando si connette, il servant, per prima cosa, annuncia la sua presenza. Il servant al quale ci si connette invia il messaggio agli altri servant ai quali è già connesso e così via finché il messaggio non si propaga sull'intera rete. Ogni servant risponde al messaggio allegando poche informazioni circa la quantità di file scaricati, etc...Così durante la connessione si può conoscere quanto materiale è disponibile sulla rete.
La GWebCache è un sistema distribuito che aiuta i servant a connettersi alla rete Gnutella1, risolvendo il problema dell'accesso alla rete, ovvero la difficoltà iniziale di conoscere l'IP di almeno un altro servant connesso alla rete.

I servant interrogano uno dei tanti server GWebCache per trovare indirizzi degli altri servant. I server GWebCache sono solitamente server Web che eseguono un software speciale.

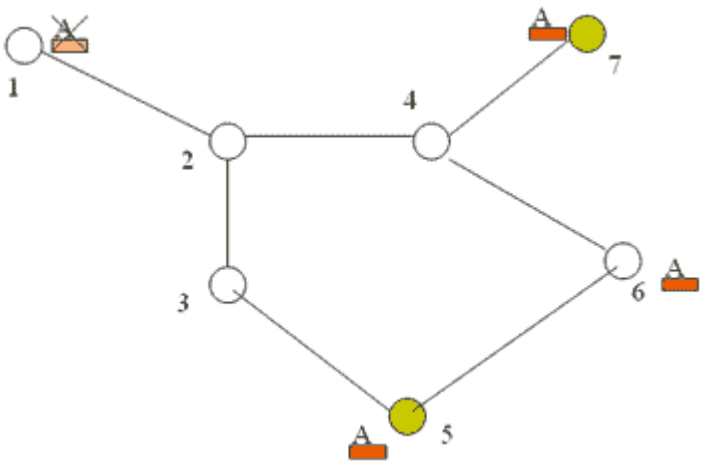
La ricerca funziona in modo simile al meccanismo di connessione: si invia una richiesta di ricerca, questa si propaga attraverso la rete ed ogni servant che possiede quel file invia i risultati indietro



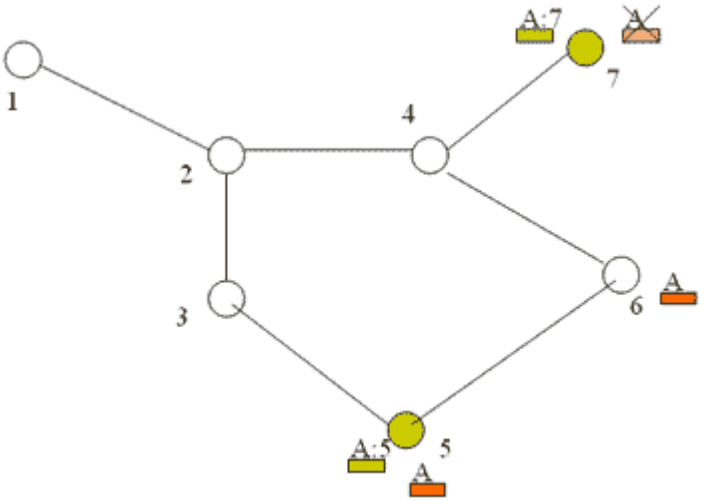
Il nodo 2 avvia la ricerca del file A



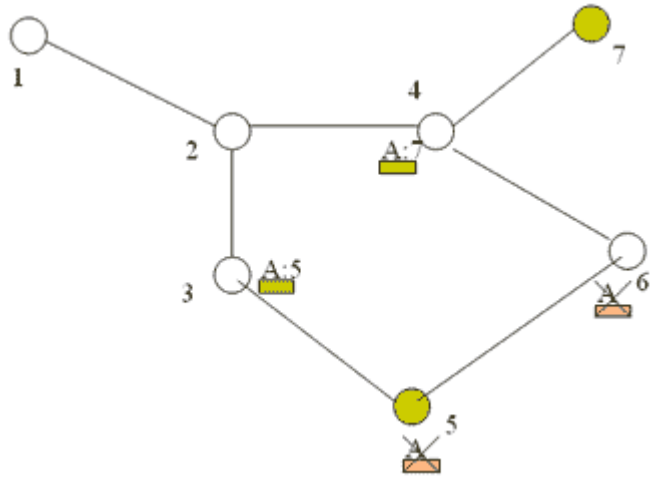
Invia il messaggio ai suoi vicini

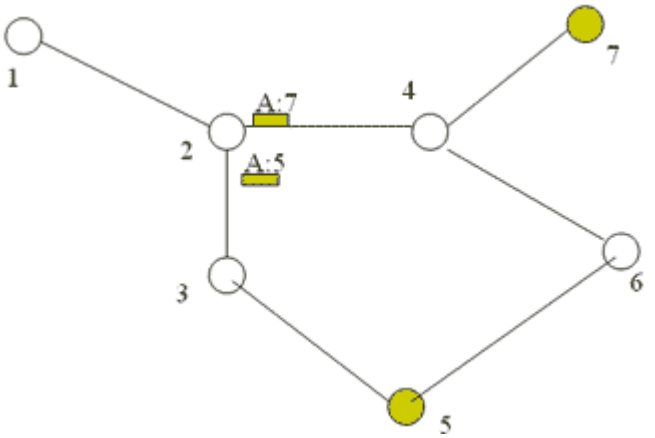


I nodi vicini propagano il messaggio ad altri nodi

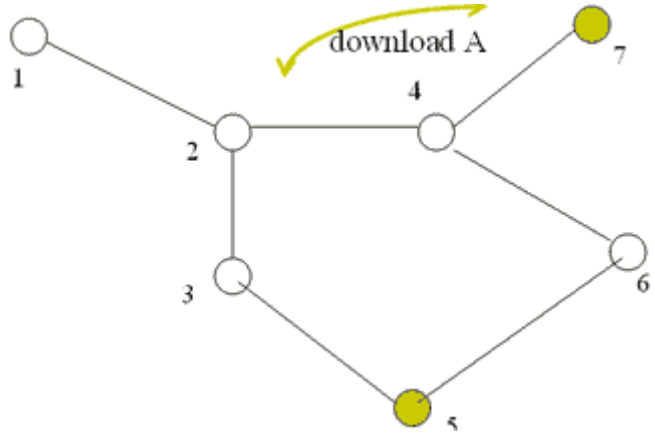


I nodi che hanno il file A creano un messaggio di risposta





I messaggi di risposta si propagano velocemente indietro



Inizia il trasferimento

Per la condivisione di file, ogni servant si comporta come un piccolo Web server http: quando trova un risultato di ricerca di un file che vuole scaricare, il nodo si connette direttamente al servant attraverso il protocollo http, così come un Web browser si connetterebbe ad un Web server.

Proprio come i pacchetti TCP/IP, i pacchetti Gnutella hanno un TTL (Time To Live). Esso serve per evitare che il pacchetto circoli nella rete per sempre. In questo modo il percorso che il pacchetto deve compiere per andare dal mittente al destinatario ha un numero massimo di passi da poter compiere. Il TTL viene inizializzato ad un numero intero. Ogni volta che un pacchetto passa attraverso un servant diverso, quest'ultimo diminuisce il valore del TTL di 1. Una volta che il TTL raggiunge il valore 0 il pacchetto non viene più propagato ad altri servant. Inoltre ogni servant ha l'opzione di ridurre arbitrariamente il valore del TTL di un pacchetto se pensa che abbia un valore irragionevole. Se, ad esempio, vengono inviati tutti i pacchetti con un TTL di 200, la maggior parte dei servant, presenti lungo il percorso, ridurranno immediatamente il valore del TTL ad un numero più ragionevole.

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 alcune limitazioni alla scalabilità dovute proprio al fatto di non aver previsto alcun modo di avvalersi della profonda eterogeneità dei nodi che la compongono e anzi subire dei rallentamenti globali a causa dei nodi lenti. Ultrapeer 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 3 mostra una tipica architettura UltraPeer. Questa soluzione prevede che i nodi nella rete non siano omogenei, ma differenziati in due 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 stesso. Per ottenere il mascheramento dei peer si utilizza lo schema Query Routing Protocol (QRP), col quale è possibile evitare di esportare i dettagli delle risorse condivise, garantendo riservatezza alle foglie. La connessione tra Supernodi, invece, mantiene il protocollo 0.6, pur essendo possibile utilizzare QRP.

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 (KaZaA), della quale però non si conoscono perfettamente le specifiche tecniche.

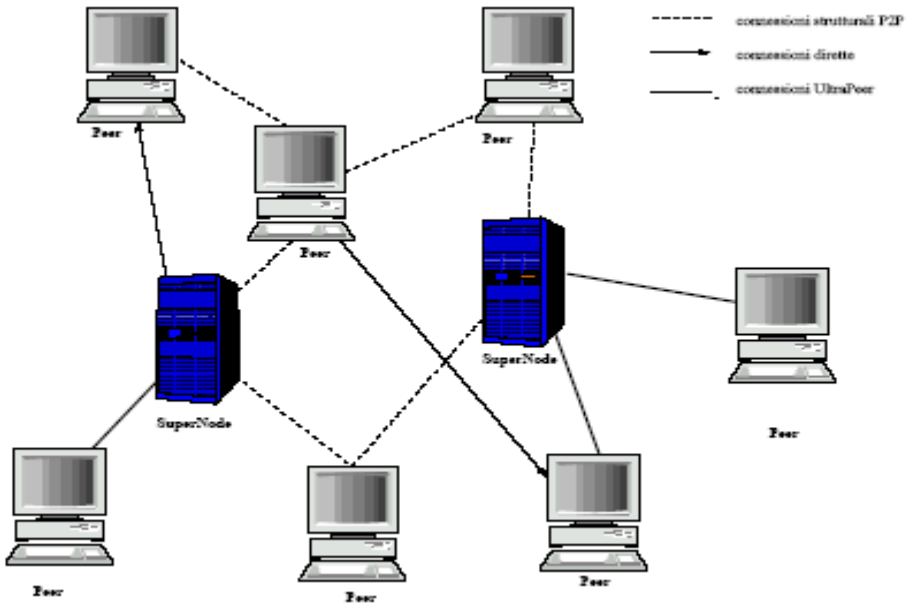


Figura 3: Architettura Gnutella1

Sono tanti i client che possono essere utilizzati per la rete Gnutella1. I principali client sono:

1) BearShare (http://www.bearshare.com): client per Windows, facile da usare e comprensivo di documentazione in italiano.
2) Gnucleus (http://sourceforge.net): client opensource per Windows.

3) GTK-Gnutella (http://sourceforge.net): client per Linux.
4) LimeWire (http://www.limewire.com): client scritto in Java, che funziona su tutte le piattaforme (Windows, Linux, Macintosh).

1.3 La rete Gnutella2

Gnutella2 è una rete peer-to-peer di terza generazione moderna ed efficiente progettata per offrire delle fondamenta solide a servizi distribuiti globali come la comunicazione punto a punto, trasferimento dati e altri servizi futuri. Essa è stata sviluppata dagli stessi realizzatori di Gnutella1. Gnutella 2 è la rete principale usata dal programma client Shareaza (http://www.shareaza.com) e da altri client compatibili.
Gnutella2 è una rete esclusiva rispetto alle attuali reti P2P sotto diversi punti di vista:

- Molte delle attuali reti di successo sono "chiuse", ovvero sono di proprietà di una singola entità con restrizioni più o meno marcate. Questo non è un modello percorribile per realizzare una rete aperta e, soprattutto, che abbia uno scopo generico. Gnutella2 ha una architettura aperta dove ognuno può partecipare e contribuire. La rete è stata progettata per consentire una tale diversità senza la necessità di comprometterne l'integrità
- La maggior parte delle reti sono dedicate esclusivamente ad uno scopo, spesso la condivisione di file. Questa è un'applicazione molto popolare per una tecnologia peer-to-peer ma non è certamente l'unica possibile. Gnutella2 è progettata come rete generica e può essere un solido punto di partenza per un gran numero di applicazioni peer-to-peer diverse, il file sharing, ma anche strumenti di comunicazione e altre idee che saranno concepite in futuro

Il nome Gnutella2 deriva dai due componenti di cui racchiude il significato: lo *Standard Gnutella 2* e la *rete Gnutella2*.
Lo **Standard Gnutella2** è un insieme di specifiche, richieste per costruire applicazioni che operino sulla rete Gnutella2 in diversi modi. Esso specifica il

minimo livello di compatibilità richiesta per essere riconosciuta come applicazione compatibile con Gnutella2.

La **Rete Gnutella2** è la componente più facilmente riconoscibile. E' una nuova architettura di rete peer-to-peer ad alta performance sulla quale possono essere costruite un gran numero di applicazioni distribuite come le applicazioni per il file sharing. Essa è una collezione auto-gestita di nodi interconnessi che cooperano per svolgere produttive attività distribuite. Non tutti i nodi che partecipano al sistema sono uguali: ci sono due principali tipi di nodi, i nodi **leaf** ("foglia") e i nodi **hub** ("centro"). Lo scopo è di massimizzare il numero di leaf e di minimizzare il numero di hub. Tuttavia, a causa della natura limitata delle risorse, il rapporto praticabile tra leaf ed hub è limitato. Si parla, perciò, di densità di leaf.

- **Nodi leaf:** sono i nodi più comuni, non hanno responsabilità speciali e non costituiscono una parte operosa dell'infrastruttura di rete. I nodi con risorse limitate devono operare come nodi leaf. Per risorse limitate si intende banda limitata, CPU o RAM inadeguati, tempo in cui si resta online, ed incapacità ad accettare connessioni TCP in ingresso.
- **Nodi hub:** costituiscono una parte importante e attiva dell'infrastruttura di rete, organizzando i nodi circostanti, filtrando ed indirizzando il traffico. I nodi hub sacrificano una parte delle loro risorse alla rete, e, quindi, la loro capacità di partecipare a funzioni di rete di alto livello è limitata. I nodi vengono eletti al grado di hub in base alle regole riportate di seguito:

  1) Sistema operativo adatto
  2) Una buona CPU e una quantità di RAM adatta
  3) Elevato tempo di permanenza online (almeno 2 ore)
  4) Banda di connessione adeguata, soprattutto la banda in ingresso
  5) Possibilità di accettare connessioni TCP in ingresso

I nodi che funzionano come hub sono fortemente interconnessi fra loro, formando così una rete di hub, dove ognuno di essi è connesso ad altri 5-30 hub vicini; il numero di interconnessioni di hub può crescere in base alle dimensioni della rete; ogni hub accetta anche connessioni da una numerosa collezione di nodi leaf, tipicamente 300-500 in relazione alle risorse disponibili. I nodi leaf si connettono simultaneamente a 3 hub e, dal punto di vista degli hub, ogni leaf viene visto come un collegamento finale.

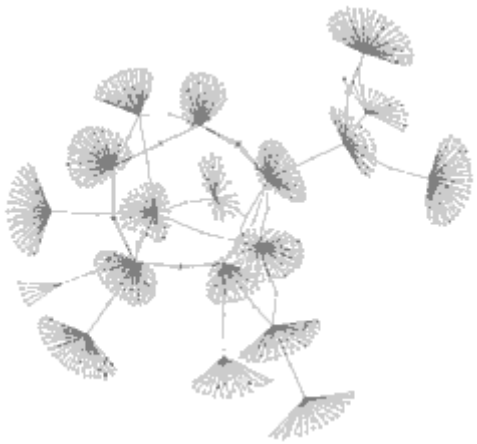


Figura 4: Rappresentazione della rete Gnutella2

Il gruppo di hub all'interno della rete, che comprende l'hub locale e i suoi vicini, viene chiamato **hub cluster** e rappresenta una modalità di raggruppamento importante. Gli hub cluster conservano una comunicazione costante con gli altri cluster, condividendo informazioni sul carico della rete e scambiando informazioni relative alla GWebCache, della quale abbiamo già parlato relativamente alla rete Gnutella1. L'hub cluster rappresenta la più piccola unità di rete in cui è possibile svolgere delle ricerche.
Le responsabilità degli hub sono:

- Mantenere aggiornate le informazioni relative agli altri hub nel cluster e ai loro hub vicini, fornendo gli aggiornamenti

- Conservare un indice delle risorse condivise da tutte le connessioni che l'hub stesso stabilisce, ovvero le risorse condivise dagli hub vicini e, soprattutto, dai leaf connessi a tale hub
- Monitorare lo stato delle connessioni locali per decidere se declassarsi allo stato di leaf, e mantenere aggiornati i servizi di connessione, ad esempio la GWebCache

Ogni nodo, hub o leaf, appena stabilisce la connessione con un altro nodo, deve dichiarare se è un nodo hub o un nodo leaf e fornire informazioni riguardo le proprie capacità.
Ogni nodo di Gnutella2 deve conservare un elenco di hub conosciuti a livello globale e un elenco completo e dettagliato degli hub che partecipano ad hub cluster adiacenti. Il meccanismo di ricerca di oggetti distribuiti sulla rete è una componente importante di Gnutella2. Consente ad un client di localizzare oggetti distribuiti sulla rete in maniera ottimale, richiedendo e ricevendo tutte le informazioni conosciute su quell'oggetto. Il meccanismo di ricerca degli oggetti su Gnutella 2 è simile a quello di Gnutella1 e avviene attraverso i seguenti passaggi:

- Un client contatta iterativamente gli hub conosciuti inviando loro la sua ricerca
- Gli hub utilizzano le parole di ricerca che hanno ricevuto comunicando con i loro vicini
- Appena viene trovata una corrispondenza, la ricerca viene propagata agli altri hub
- La ricerca si estende almeno ad un cluster, che è l'unità base di ricerca.
- I nodi, che ricevono la richiesta di ricerca filtrata, la processano ed inviano i risultati direttamente al client che aveva avviato tale ricerca

1.4 FreeNet

FreeNet [2] è una rete peer-to-peer decisamente diversa da tante altre in quanto prevede un alto grado di riservatezza ed anonimato garantito agli utenti. Si può definire come un sistema distribuito per la memorizzazione di informazioni, realizzato per proteggere la riservatezza degli utenti e preservare la sopravvivenza della rete stessa.

Esso funziona come una rete punto a punto autogestita, che mette in condivisione delle spazio disco tra centinaia di migliaia di computer per creare un file system virtuale collaborativo. Questa è una prima grande differenza con gli altri sistemi P2P. Ad esempio, quando decidiamo in FreeNet di condividere sulla rete un file video di 300 Megabytes non lo condividiamo conservandolo sul nostro hard disk, ma FreeNet lo dividerà in tante piccole parti che invierà sulla rete distribuendole secondo algoritmi intelligenti sullo spazio disco di centinaia di altri nodi. Lo spazio disco, che noi mettiamo a disposizione della rete, viene utilizzato per la memorizzazione di altri file che la rete stessa sceglie a seconda della necessità e di cui noi non conosciamo l'identità, perché tali file vengono memorizzati crittografati.

Al fine di aumentare la robustezza della rete ed eliminarne i punti deboli, FreeNet utilizza un'architettura completamente decentralizzata ed implementa delle strategie per proteggere l'integrità dei dati e prevenire problemi di privacy. Il sistema è anche capace di replicare o cancellare file autonomamente sulla rete al fine di massimizzare l'efficienza di utilizzo dello spazio disco complessivo disponibile in risposta alla domanda.

Nel progettare FreeNet [10] ci si è concentrati sui seguenti punti :

1) conservare l'anonimato di chi produce dell'informazione (condivide un articolo, un file, etc…), di chi la richiede e di chi la trasporta sulla rete
2) alta disponibilità ed affidabilità dell'informazione attraverso un meccanismo decentralizzato
3) sistema efficiente per la memorizzazione ed l'instradamento dei dati
4) resistenza alla censura

Gli sviluppatori pensano che sia inutile proteggere la privacy per la creazione ed il download dei file se non si proteggono i file stessi, in particolare, se non si evita che chi trasporta i file sulla rete non sia soggetto ad attacchi o investigazioni di ogni genere. Per questo motivo FreeNet rende difficilissimo capire esattamente quale computer trasporta un determinato file. Il fatto che non si conosca dove sia memorizzato effettivamente un dato file e un meccanismo di replicazione ridondante dei dati rende estremamente difficile per i censori eliminare i file presenti nella rete.

FreeNet, tuttavia, non garantisce una memorizzazione permanente dei dati sulla rete. Siccome lo spazio disco è finito, esiste un compromesso tra pubblicare nuovi file e conservare quelli vecchi: per evitare che dati inutili occupino tutto lo spazio provocando l'automatica cancellazione dei dati esistenti, FreeNet implementa una politica di memorizzazione probabilistica. Gli autori, ovviamente, sperano che FreeNet attragga sufficienti risorse da parte dei partecipanti in modo da conservare indefinitamente la maggior parte dei file.

Ogni utente di FreeNet è un nodo che offre alla rete una desiderata quantità di spazio disco. Per aggiungere un nuovo file alla rete l'utente invia un messaggio di inserimento contenente il file ed il suo identificatore indipendente dalla locazione in cui tale file verrà memorizzato. Questo identificatore si chiama GUID (Globally Unique IDentifier). Il file verrà memorizzato su un insieme di nodi. Durante tutto il tempo in cui il file resterà sulla rete (ovvero fino a quando non verrà automaticamente cancellato perché nessuno lo richiede più), esso potrà migrare su un altro insieme di nodi oppure essere duplicato. Per scaricare quel file un altro utente dovrà inviare alla rete un messaggio di download contenente il GUID del file. Quando la richiesta raggiunge uno dei nodi dove è memorizzato quel file (o parte di quel file), i dati vengono inviati indietro lungo la catena verso chi aveva inoltrato la richiesta.

Le chiavi GUID vengono calcolate usando l'algoritmo sicuro SHA-1. La rete impiega tre tipi principali di chiavi: la keyword-signed key (KSK), il content-hash key (CHK), usato per la memorizzazione dei dati, ed il signed-subspace key (SSK), pensato per un uso di più alto livello.

- **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 da comunicare. Presenta, tuttavia, un inconveniente: essendoci un unico "spazio nome" è possibile che due file differenti abbiano la stessa descrizione, e, quindi, lo stesso nome, rendendoli entrambi irreperibili
- **Content-Hash Key (CHK):** è la chiave che viene usata per la memorizzazione dei dati a basso livello e viene generata calcolando un codice hash in base ai contenuti. Questo processo fornisce ad ogni file un identificatore assolutamente unico (collisioni SHA-1 vengono considerate quasi impossibili), che può essere verificato molto velocemente. I CHK permettono che differenti copie dello stesso file inserite da persone diverse sulla rete vengano automaticamente unite
- **Signed-Subspace Key (SSK):** è la chiave che crea uno spazio per il nome del file che tutti possono leggere, ma che solo il proprietario originario di quel file (ovvero che lo ha immesso sulla rete per la prima volta) può modificare. Firmare il file con la chiave privata consente di verificare l'integrità del file, perché ogni client, che entra in possesso della SSK, può verificarne la firma prima di accettare il download

Quindi:

1) Per scaricare un file occorre avere la chiave pubblica e la stringa descrittiva attribuita al file (nome). Con queste due informazioni è possibile ricreare la chiave SSK
2) Per aggiornare un file occorre anche la chiave privata per generare una firma valida. Questo assicura che solo il proprietario di quel file possa modificarlo successivamente al rilascio sulla rete

FreeNet è stata progettata sin dall'inizio per affrontare con successo attacchi ostili dall'interno e dall'esterno volti a minare la privacy dei suoi utenti. Perciò rende intenzionalmente complesso il modo con cui i vari nodi direzionano i messaggi tra di loro e rende l'instradamento dei messaggi dinamico e nascosto.

In FreeNet i messaggi invece di viaggiare direttamente dal mittente al destinatario, viaggiano nodo-a-nodo. In tali catene ogni collegamento è criptato, finché il messaggio raggiunge il destinatario. Siccome ogni nodo della catena conosce solo i suoi vicini, i punti terminali di questa catena (mittente e destinatario) potrebbero trovarsi ovunque tra migliaia di nodi, che scambiano in continuazione messaggi indecifrabili. Nemmeno il nodo immediatamente vicino al mittente può sapere e stabilire se il suo predecessore è stato davvero lui a generare quel messaggio oppure se lo sta semplicemente facendo transitare sulla rete da un nodo all'altro lungo la catena di nodi. Allo stesso modo il nodo che precede il destinatario non può dire ne stabilire se il suo successore sarà proprio il destinatario o un semplice nodo che continuerà a far transitare il messaggio.

L'instradamento delle richieste verso i dati è la parte più importante del sistema FreeNet. Il metodo più semplice di instradamento, usato da servizi come Napster, consiste nel mantenere un indice centrale di file; in questo modo gli utenti possono inviare le richieste direttamente a chi possiede i file che cercano. Sfortunatamente la centralizzazione crea un unico grande punto debole, che è facile da attaccare.

Ogni nodo mantiene un tabella di instradamento (routing table), che elenca gli indirizzi degli altri nodi e le chiavi GUID, che, ritiene, tali nodi possano avere. Quando un nodo riceve una richiesta, analizza per prima cosa il suo spazio disco per vedere se, eventualmente, trova lì il file. Se lo trova allora lo invia indietro al nodo dal quale ha ricevuto la richiesta affiancandoci un tag, che lo identifica come possessore di quel file. Altrimenti il nodo invia la richiesta al nodo che nella sua tabella ha la chiave più simile a quella cercata. Se la richiesta ha successo ogni nodo nella catena lascia passare il file indietro verso il destinatario. Inoltre ogni nodo potrebbe anche conservare sul proprio spazio una copia di quel file.

Per nascondere l'identità del possessore dei dati i nodi occasionalmente alterano i

messaggi di risposta, fingendo di essere loro i possessori del file. Con richieste successive si riuscirà di nuovo a trovare i dati perché i nodi conservano la vera identità del possessore del file nella loro tabella di instradamento e, quindi, si riuscirà a indirizzare la richiesta nella direzione giusta. Le tabelle di instradamento non vengono mai rivelate agli altri nodi.

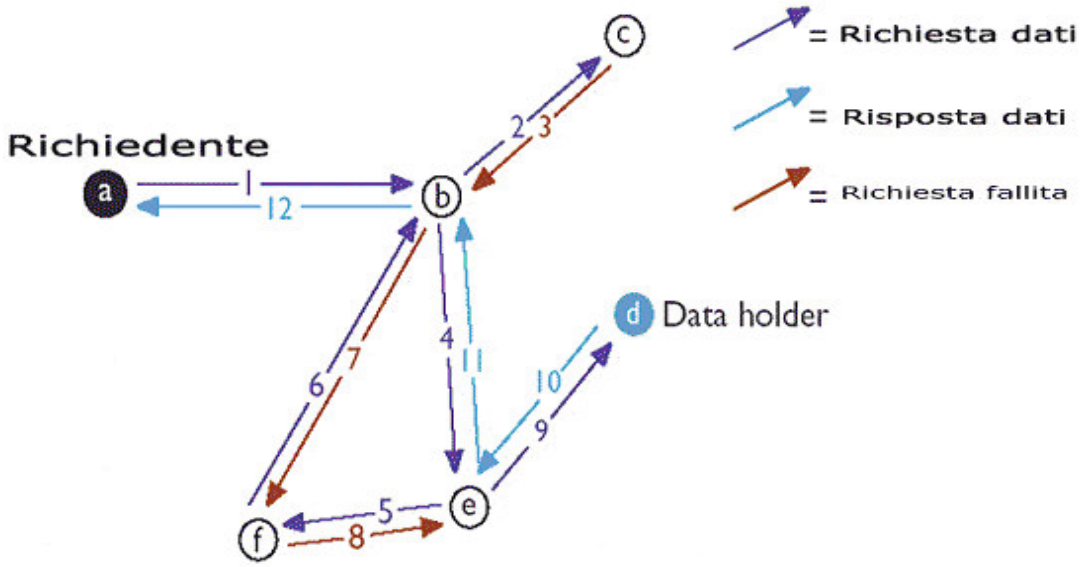


Figura 5: Sequenza di ricerca in FreeNet

In Figura 5 è rappresentata una tipica sequenza di ricerca. L'utente avvia la ricerca al nodo A e la spedisce a B, il quale la spedisce a C. Il nodo C non è capace di contattare altri nodi e risponde a B con un messaggio di "richiesta fallita". Il nodo B allora prova con la sua seconda scelta, E , che invia il messaggio al nodo F. Il nodo F a sua volta invia il messaggio a B che rileva un loop (ovvero arrivo di un messaggio di richiesta ad un nodo dal quale quel messaggio è già passato) e rispedisce il messaggio indietro. Incapace di contattare altri nodi, il nodo F rispedisce ancora indietro il messaggio ad E, che spedisce il messaggio di richiesta alla sua seconda scelta nella tabella di instradamaneto, D. Quest'ultimo trova il file sul suo hard disk. A questo punto D invia il file indietro attraverso E , quindi a B fino a farlo giungere al nodo A dal quale era partita la richiesta. Lungo il percorso, E, B ed A possono anche conservare una copia nel proprio spazio disco di questo file.

Con questo meccanismo una ricerca successiva dello stesso file tenderà a seguire lo stesso percorso ed una copia del file salvata localmente potrà soddisfare la richiesta prima di quella precedente.

I nodi che rispondono in maniera positiva alle richieste saranno aggiunti a più tabelle di instradamento e, quindi, saranno contattati più spesso degli altri.

Per connettersi alla rete un nuovo nodo genera per prima cosa un coppia di chiavi pubblica-privata per se stesso. Questa coppia serve per identificare logicamente quel nodo e viene usata per verificare l'indirizzo fisico di riferimento. A questo punto il nodo invia un messaggio di annuncio, comprensivo della chiave pubblica ed del suo reale indirizzo fisico, ad un nodo esistente che può individuare in diversi modi come, ad esempio, attraverso gli elenchi di nodi presenti nel Web. Il nodo che riceve l'annuncio, si annota le informazioni identificative del nuovo nodo ed invia l'annuncio anche ad altri nodi scelti in maniera casuale nella sua tabella di instradamento. L'annuncio continua a propagarsi.

Man mano che vengono processate più richieste, l'instradamento della rete dovrebbe diventare più performante. Le tabelle di instradamento dei nodi dovrebbero specializzarsi a gestire gruppi di chiavi simili, poiché ogni nodo riceverà soprattutto richieste di chiavi che sono simili alle chiavi alle quali è associato nelle tabelle di instradamento degli altri nodi. Quando quelle richieste hanno successo i nodi imparano qualcosa su nodi precedentemente sconosciuti che possono rispondere positivamente a determinate chiavi e creano per loro nuovi campi nelle tabelle di instradamento.

Per incoraggiare la partecipazione, FreeNet non impone nessun limite circa la quantità di dati che è possibile inserire nella rete. Poiché però lo spazio disco a disposizione è finito, il sistema di tanto in tanto deve decidere quali file mantenere e quali eliminare. Attualmente, nell'allocare lo spazio disco, si dà priorità ai file più popolari, ovvero quelli che vengono richiesti più frequentemente. Ogni nodo ordina i file nel proprio spazio disco in base alla data dell'ultima richiesta, e quando arriva un nuovo file che non può essere inserito perché non c'è più spazio disponibile, il nodo cancella i file meno richiesti per fare spazio.

Le informazioni nelle tabelle di instradamento vengono mantenute anche dopo la cancellazione dei file. Il nodo, in questo modo, potrà rispondere a successive richieste per quel file che ha cancellato usando la sua tabella di instradamento per indirizzare la richiesta verso il nodo che ha distribuito per primo quel file e che potrebbe essere in grado di servirne attualmente una copia. La distribuzione dei file è perciò determinata da due fattori dominanti: crescita delle domanda e cancellazione file.

1.5 Napster

Napster [6] 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 un vero e proprio fenomeno di massa. Napster è in grado di facilitare lo scambio di file musicali MP3 tra gli utenti di tutto il mondo. Perché il sistema funzioni, tutti gli utenti devono scaricare un client che, collegandosi ai server di Napster, permette 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 crea un canale di scambio in rete che permette 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 questo, fino a pochi anni fa, avveniva gratuitamente.

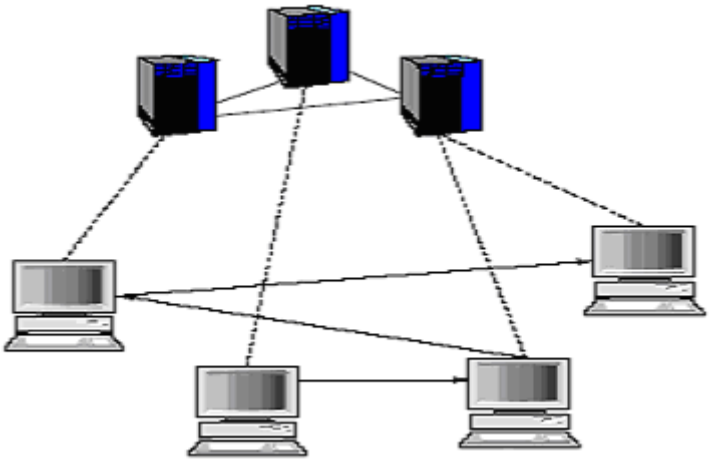


Figura 6: Architettura di Napster

L'architettura di Napster ha una struttura ibrida centralizzata, come illustrato in Figura 6. 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 vengono 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 viene descritto dal percorso di accesso sul disco rigido dell'utente, dal nome, dalla dimensione e dalla qualità. Ci sono due modi per effettuare le ricerche: il primo tipo di interrogazione prevede parole chiave, limitazioni sulle dimensioni especifiche di qualità e fornisce, come risultato, l'elenco delle risorse, che soddisfano le richieste. Per ognuno di questi risultati viene elencato: login dell'utente, indirizzo IP e porta, velocità di connessione e i dettagli del file condiviso. Effettuata la scelta, viene instaurata una connessione diretta, che coinvolge i due peer, che, a questo punto, scambiano la risorsa in maniera autonoma. L'altro tipo di interrogazione, invece, prevede la possibilità di scorrere l'elenco dei file messi a disposizione di ogni singolo utente, conoscendone lo pseudonimo.

A questi sistemi è 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. Una strategia, che viene spesso praticata dagli utenti, è 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 porta 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.

L’anno scorso, la partnership tra il colosso discografico Bertelsmann e Napster ha dato come risultato che, l'accesso a Napster non è più gratuito, ma si deve pagare un canone mensile per aver diritto all'abbonamento.

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 [7]. 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 (http://www.winmx.com), che sfrutta una topologia ibrida decentralizzata a cluster per offrire risorse multimediali.

1.6 KaZaA

Una piccola rivoluzione è stata portata da **KaZaA** [4] che, in collaborazione con **FastTrack** [1], ha realizzato un sistema basato su architettura ibrida a Supernodi estremamente efficiente e funzionale.

Il protocollo FastTrack è stato inventato dagli scandinavi Niklas Zennström e Janus Friis e fu rilasciato per la prima volta nel Marzo 2001 dalla loro ditta tedesco-olandese Consumer Empowerment. Attualmente i diritti di KaZaA sono esercitati dalla Sharman Networks.

KaZaA Media Desktop è un'applicazione peer-to-peer per il file sharing che utilizza il protocollo FastTrack; viene principalmente utilizzata per scambiare file musicali mp3 e video, e dal 2003 è diventata l'applicazione peer-to-peer più diffusa nel mondo. Ogni giorno più di 3 milioni di utenti utilizzano KaZaA per condividere più di 5000 terabyte di informazioni (mp3, video, immagini,…). Il programma viene rilasciato attualmente solo per il sistema operativo Windows. È possibile eseguirlo anche su Linux o Mac OS X o altri sistemi operativi, ma è necessario dotarsi di un software di emulazione come WINE e Virtual PC.

KaZaA è molto simile agli altri client Morpheus e Grokster, in quanto utilizza lo stesso protocollo ed ha una tecnologia peer-to-peer decentralizzata simile a quella di

Gnutella. Molti considerano KaZaA superiore ad altri programmi di file sharing grazie alla grandissima scelta di file e alla sua velocità di trasferimento, altri invece fanno notare che il client KaZaA installa degli spyware sui personal computer degli utenti. Viene, infatti, installato, di default, il software Altanet che crea problemi, come l'allocazione di una certa ampiezza di banda per inviare pubblicità. Questo programma consente anche il prelievo di musica a pagamento, creando una alternativa legale allo scambio di file musicali coperti da copyright.

Nell'Agosto del 2003 venne rilasciata **KaZaA Plus**. Questa versione a pagamento non aveva né spyware né adware ed è stata anche creata una versione commerciale chiamata **KaZaA Gold**. Esiste un'altra versione di KaZaA chiamata **KaZaA Lite,** o **K-Lite**, che è una modifica alla versione di KaZaA, esclude tutti gli spyware e adware, fornisce qualche funzionalità in più e può essere scaricata liberamente. Si collega anch'essa alla rete FastTrack consentendo lo scambio di file con gli utenti di KaZaA. È stata realizzata da altri programmatori modificando il file binario dell'applicazione KaZaA originale. La Sharman Network ha considerato l'applicazione KaZaA Lite come una violazione del copyright e l'11 Agosto 2003 inviarono una lettera a Google chiedendo che tutti i link all'applicazione K-Lite venissero rimossi dal loro database. Nel Dicembre 2003 Sharman minacciò di adire per vie legali contro i proprietari di tutti i siti che ospitavano una copia di K-Lite, a meno che non la rimuovessero. Attualmente non è più presente sul suo sito ufficiale, è difficile trovarla, ma da qualche parte nel Web si trova ancora. L'aspetto curioso è che il programma è straordinariamente disponibile sulla rete stessa di FastTrack, e che può essere facilmente scaricato con KaZaA o con altri client FastTrack. Probabilmente la nuova versione di KaZaA impedirà agli utenti di K-lite di collegarsi ai loro supernodi, nello sforzo di chiudere a questi utenti l'accesso sulle reti FastTrack.

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:

1) **SmartStream**: offre un rimedio ad un tipico problema di Napster: la difficoltà di 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
2) **FastStream**: 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

Lo stack peer-to-peer di FastTrack 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 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 servant. 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.

Come altri proprietari di prodotti simili, i creatori di KaZaA sono stati citati a giudizio dalle major discografiche al fine di limitare lo scambio e la condivisione di materiale coperto da copyright. La magistratura olandese ordinò ai proprietari di KaZaA di impedire ai loro utenti la violazione del copyright, altrimenti avrebbero dovuto pagare una pesante multa. I proprietari di KaZaA risposero vendendo l'applicazione alla Sharman Networks con quartier generale in Australia e con una filiale nell'isola di Vanuatu. Verso la fine di Marzo del 2002 la corte di appello olandese capovolse il precedente giudizio affermando che KaZaA non poteva essere responsabile delle azioni dei suoi utenti. Nel 2002 la Sharman fu citata in giudizio dalla RIAA e dalla MPAA. Il processo è ancora in corso. La Sharman rispose attraverso una azione legale antitrust, sostenendo che le major avevano cospirato contro la Sharman a causa del servizio a pagamento di Altanet. Nel Luglio 2003 la causa finì. La Sharman inoltre sostenne che non poteva essere citata in giudizio in California in quanto non aveva sufficienti relazioni con quello stato. Nel Settembre 2003 la RIAA presentò una richiesta di citazione a giudizio alla corte civile contro diversi utenti privati che avevano condiviso una grande quantità di file con KaZaA . Questi furono costretti a pagare alla RIAA un piccolo risarcimento. Come risultato il traffico sulla rete di FastTrack subì una contrazione di circa il 10-15%. La Sharman rispose citando a giudizio la RIAA, affermano che la propria rete era stata violata da altri software client (come KaZaA Lite) usati per investigare chi stava condividendo file, sulla rete di KaZaA. Nel Gennaio 2004 la corte rigettò le accuse dalla Sharman.

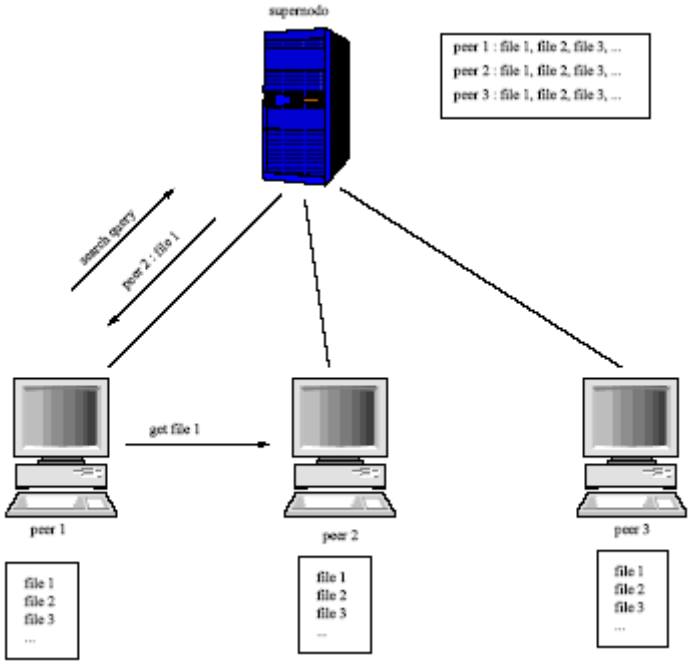


Figura 7: Architettura di KaZaA

Siccome FastTrack è un protocollo proprietario che cripta i suoi messaggi per l'invio dei segnali, poco si conosce sulle specifiche della Overlay Network di KaZaA, sulla gestione di tale rete, e sul protocollo dei segnali. Nel Settembre 2004, **Jian Liang** e **Keith W. Ross**, del Dipartimento di Computer and Information Science Polytechnic e **Rakesh Kumar**, del Dipartimento di Electrical and Computer Engeneering Polytechnic dell'Università di Brooklyn, hanno effettuato uno studio di misurazione [11] sulla struttura e dinamica della rete di KaZaA, sulla sua selezione del vicino, sul suo uso dinamico dei numeri di porta per eludere i firewall, permettendo di descrivere KaZaA e la sua rete e cercando di attivare nei lettori delle intuizioni sui dettagli del programma P2P.

Per cercare di svelare i misteri della rete di KaZaA, sono stati realizzati due dispositivi di misurazione: il **KaZaA Sniffing Platform** e il **KaZaA Overlay Probing Tool**. Il primo consiste di un insieme di nodi che vengono fatti interconnettere fra di loro in maniera controllata, e ognuno di loro, inoltre, è connesso a tante piattaforme esterne di nodi KaZaA. Esso viene utilizzato per conoscere il

traffico dei segnali della rete. Il secondo stabilisce una connessione TCP, effettua l'handshake, invia e riceve i messaggi criptati di KaZaA con ogni nodo fornitogli. Esso viene, quindi, utilizzato per analizzare la disponibilità dei nodi e per capire come viene effettuata la scelta del nodo vicino.

1.6.1 Descrizione della rete

Nella rete, i nodi differiscono per disponibilità, per larghezza di banda durante la connessione e per potenza della CPU. KaZaA è stato una dei primi sistemi P2P ad organizzare i peer in due classi, i SuperNodi (SN) e i Nodi Ordinari (ON). I SN sono generalmente più potenti in termini di connettività, larghezza di banda, e hanno anche maggiori responsabilità. Come mostrato in Figura 8, ogni ON ha un SN padre. Quando un ON lancia l'applicazione di KaZaA, l'ON sceglie un SN padre, mantiene con esso una semi-permanente connessione TCP, e trasferisce ad esso i metadati per i file che sta condividendo. KaZaA mantiene una file indice che mappa gli identificativi dei file agli indirizzi IP. Inoltre ogni SN mantiene un indice locale per tutti i suoi ON figli. In questo modo il SN è simile ad un hub di Napster, ma non è un server dedicato.

I metadati, che il ON trasferisce al suo SN padre, includono: il nome e la dimensione del file, il ContentHash, e i file descriptor, che vengono utilizzati come parole chiave durante la fase di query. Il ContentHash svolge un ruolo importante nella architettura di KaZaA: esso serve per identificare un file nella richiesta HTTP di download. Se fallisce lo scaricamento da uno specifico peer, il ContentHash permette al client di cercare automaticamente il file specifico fra gli altri peer, senza effettuare un'altra query con la stessa parola chiave.

Quando un utente vuole localizzare un file, il ON invia una query con le parole chiave al SN padre attraverso la connessione TCP. Per ogni corrispondenza nel suo database, il SN fornisce l'indirizzo IP, il numero di porta del server, e i metadati corrispondenti. Come mostrato nella Figura 8, ogni SN mantiene una connessione TCP con gli altri SN, creando una Overlay Network. Perciò, quando un SN riceve una query, esso può passare la richiesta ad uno o a più SN ai quali è connesso.

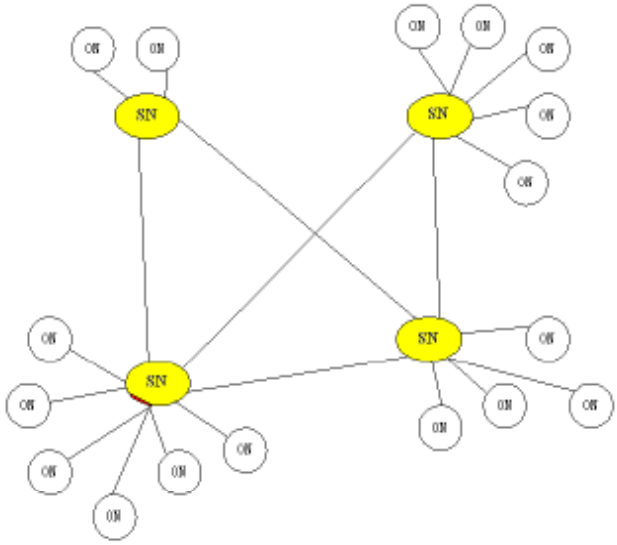


Figura 8: Overlay network a due livelli di KaZaA

Il progetto FastTrack File Format ha determinato la sintassi e la semantica dei file di sistema di KaZaA. Grazie a tale progetto, sappiamo che ogni peer di KaZaA possiede i seguenti componenti software:

1) il KaZaA Media Desktop (KMD)
2) informazione sull'ambiente software memorizzate nel Registro di Windows. Inclusa in tale informazione, c'è una lista di massimo 200 SN, che viene chiamato **SN refresh list**. Per ognuno dei SN, la lista prevede una serie di attributi, fra i quali l'indirizzo IP e il numero di porta
3) i file DBB, i quali contengono i metadati per i file che il peer vuole condividere
4) i file DAT, i quali contengono la parte di file che è stata, fino a quel momento, scaricata. Il file DAT cresce in dimensione durante il download. Una volta che il file richiesto è stato completamente scaricato, il file DAT viene rinominato con il nome effettivo del file.

Ogni peer scambia 4 tipi differenti di traffico TCP con gli altri peer nella rete:

- Traffico di segnali, che include l'handshake per stabilire la connessione tra i peer e la lista dei SN. Tale traffico è criptato
- Traffico di trasferimento dei file (mp3, video,…), trasferiti direttamente fra i peer senza il passaggio attraverso un SN intermediario. I trasferimenti dei file non sono criptati e sono inviati entro messaggi HTTP
- Annunci commerciali, inviati attraverso messaggi HTTP
- Traffico dei messaggi istantanei (instant message)

Come abbiamo detto, i messaggi per i segnali, sia SN-SN che ON-SN, sono criptati. Il progetto giFT attraverso un reverse-engineering, ha compreso gli algoritmi di criptaggio di KaZaA. Grazie a questo, gli utenti di giFT-FastTrack possono cercare e scaricare i file della rete di KaZaA. I due dispositivi di misurazione, descritti precedentemente, contengono il codice di criptaggio/decriptaggio realizzato dal progetto giFT.

Molti utenti utilizzano KaZaA-Lite, di cui abbiamo già parlato in precedenza, piuttosto che il client di KaZaA (KMD) distribuito da Sharman. Ogni client di KaZaA-Lite emula il KMD e partecipa alla rete di KaZaA. Durante il processo di ricerca, come prima cosa, un ON di KaZaA-Lite invia la sua query al SN al quale è connesso. Dopo aver ricevuto tutte le risposte dal suo SN padre, il ON si disconnette e si connette con un altro SN, e rimanda la richiesta al nuovo SN. In particolare, il ON rimanda i suoi metadati al nuovo SN e il precedente SN cancella i metadati del ON. Durante una specifica richiesta, il ON si può connettere a molti SN.

Per essere più precisi, ricordiamo che KaZaA è solo uno dei diversi client che utilizzano il protocollo FastTrack e partecipano all'Overlay di FastTrack. Oltre a KaZaA, ci sono Grockster e iMesh. Tutti e tre i client utilizzano lo stesso protocollo. Quindi, quando parliamo di KaZaA, ci riferiamo alla rete FastTrack.

1.6.2 KaZaA Sniffing Platform

Il dispositivo è costituito da tre workstation [11], ognuna con una versione di KMD 2.0 installata.

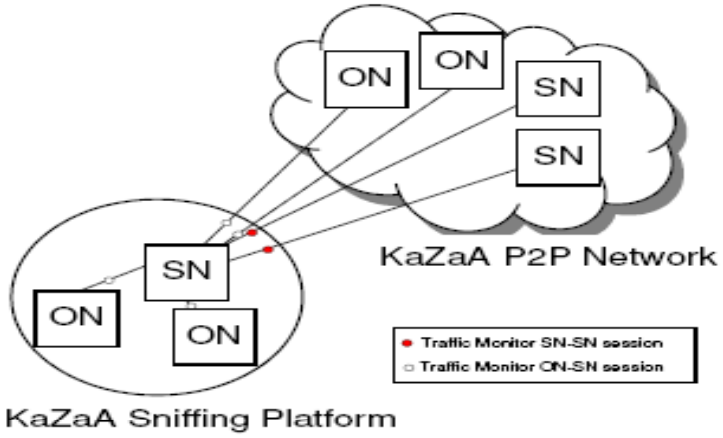


Figura 9: KaZaA Sniffing Platform

All'inizio le workstation funzionavano come ON nella rete di KaZaA e si è atteso che uno dei tre nodi venisse promosso come SN. Dopodiché sono stati manipolati i Registri di Windows negli altri due nodi in modo tale da costringerli ad usare il terzo nodo come loro SN padre.

Come si può notare dalla Figura 9, il SN si connette anche ai ON e al SN della rete di KaZaA. E' stata disposta, intorno al SN di riferimento, una serie di monitor, perché catturassero tutti i segnali in ingresso e in uscita.

La piattaforma è stata installata in due differenti sottoreti: una connessa alla rete del campus della Polytechnic University, l'altra connessa ad una rete cable residenziale. Sono state scelte queste due postazioni perché rappresentano la tipologia di sottoreti che la maggioranza dei peer di KaZaA utilizzano.

Solo in seguito è stata effettuata un'analisi offline dei dati ottenuti, determinando, così, che i nodi frequentemente si scambiano fra di loro le liste dei SN. In particolare, quando un ON si connette con il SN padre, quest'ultimo immediatamente fornisce al ON una SN refresh list, che prevede gli indirizzi IP, i numeri di porta e il carico di lavoro (Workload) di, al massimo, 200 SN. La prima riga della lista contiene i dati relativi al SN padre. Quando un ON riceve la SN refresh list dal suo SN padre, esso

solitamente elimina alcuni SN dalla sua SN refresh list e vi aggiunge altri nuovi SN appena "conosciuti". In questo modo i nodi mantengono una lista aggiornata dei SN attivi. Anche i SN vicini nella Overlay Network si scambiano le SN refresh list.

1.6.3 KaZaA Overlay Probing Tool

Quando un peer lancia il client di KaZaA [11], la prima azione di quest'ultimo è di scegliere un SN padre e di stabilire un Overlay link, ad esempio la connessione TCP, con lo stesso SN. Abbiamo scoperto che vengono seguiti i seguenti passi:

1) Come affermato in precedenza, il ON ha una SN refresh list. Esso sceglie alcuni, solitamente 5, SN candidati dalla lista e verifica che se il SN è attivo inviandogli un pacchetto UDP. In seguito il ON riceve le risposte UDP da un sottoinsieme dei candidati
2) Con ognuno dei SN, dai quali riceve risposta, il ON tenta di stabilire una connessione TCP. Per ognuna di queste connessioni, il SN e il ON si scambiano messaggi criptati, il ON manda informazioni sul peer (indirizzo IP locale, numero di porta e username), il SN manda la SN refresh list
3) Il ON sceglie uno dei SN e si disconnette dagli altri; l'unico rimasto diventa il SN padre.
4) Il ON può, quindi, effettuare la query al SN scelto.

I diagrammi per l'instaurazione delle connessioni ON-SN e SN-SN sono mostrati nella Figura 10. Come si può notare, le due procedure di instaurazione di connessione sono diverse.

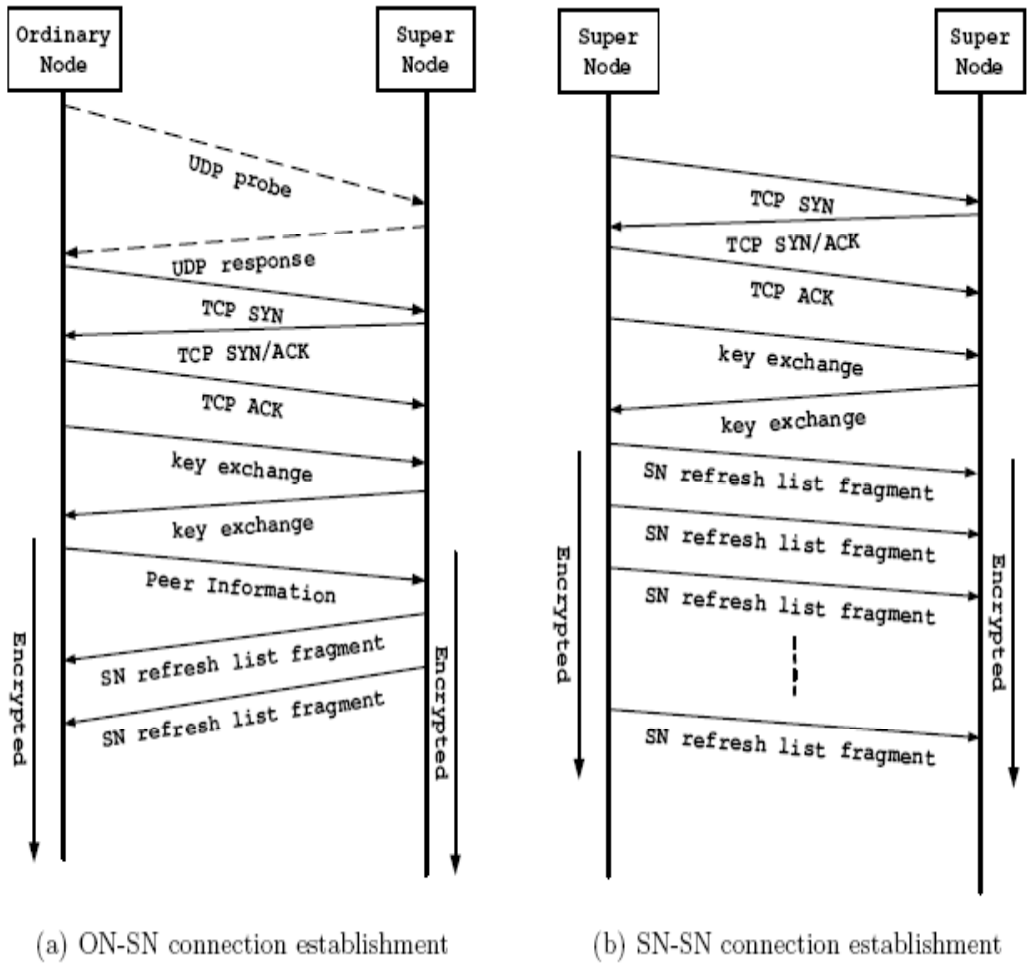


Figura 10: Instaurazione della connessione fra peer nella KaZaA Overlay

E' stata, inoltre, determinata [11] la struttura dei segnali scambiati fra i peer di KaZaA. Come mostrato in Figura 11, ogni messaggio inizia con l'identificativo "K" di un byte, seguito dal campo, di due byte, che specifica il tipo di messaggio, poi il campo, di due byte, che indica la lunghezza dell'informazione, e, infine, l'informazione stessa di lunghezza variabile.

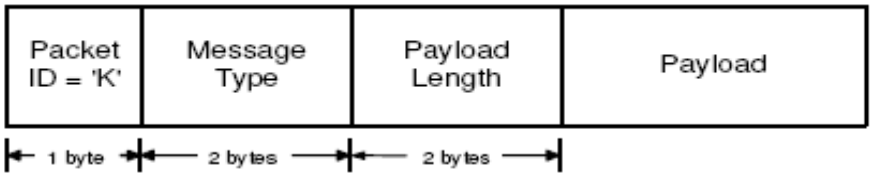
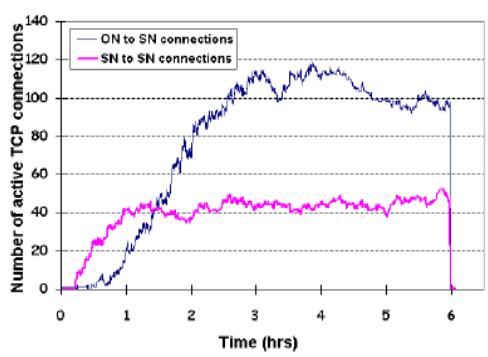


Figura 11: Formato dei segnali della rete di KaZaA

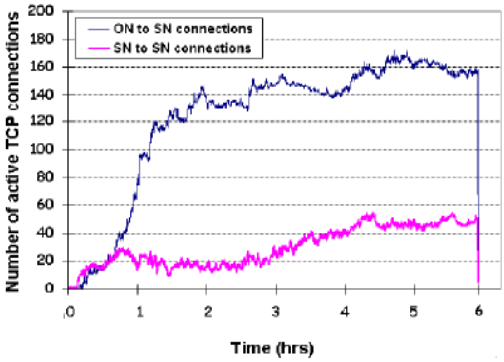
Dopo aver compreso il protocollo dei segnali e la struttura dei messaggi che circolano nella rete di KaZaA, è stato sviluppato il KaZaA Overlay Probing Tool. Questo dispositivo può completamente emulare il comportamento del client KMD per quanto riguarda l'instaurazione di nuove connessioni e lo scambio dei segnali fra i nodi. Questo tool viene utilizzato per verificare se, nella rete, ogni SN, arbitrariamente specificato, è in grado di recuperare la SN refresh list, inviata da un altro SN, e di ottenere il carico di lavoro di un SN scelto.

1.6.4 Struttura e dinamica della rete Overlay

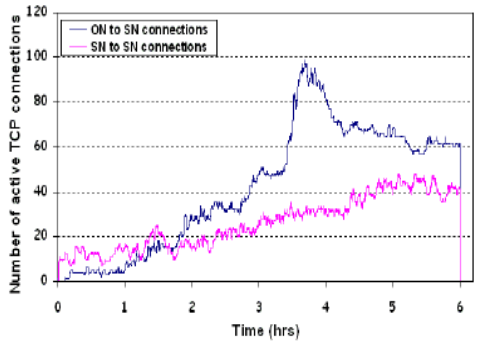
E' stata studiata [11] la connettività di un tipico SN e, attraverso il KaZaA Sniffing Platform, è stato monitorato il numero di connessioni TCP provenienti dal SN. Questo esperimento è stato effettuato in due ambienti diversi, la rete del campus del Polytechnic e una rete ad accesso residenziale, in periodi diversi.



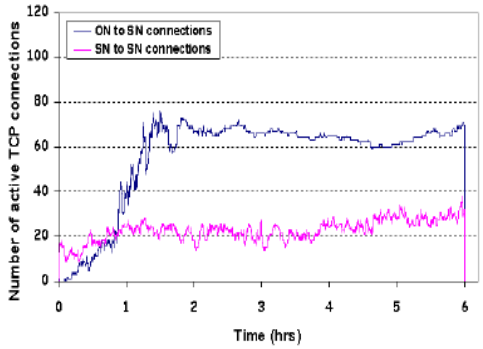
(a) Polytechnic campus - session evolution, Aug. 22, 2003



(b) Polytechnic campus - session evolution, Oct 24, 2003



(c) Residential Cable Modem - session evolution, Aug. 23, 2003



(d) Residential Cable Modem - session evolution, Oct 25, 2003

Figura 12: Evoluzione nel tempo delle connessioni SN-SN e ON-SN

La Figura 12 mostra i risultati: il numero di connessioni inizia da uno e sale fino ad un valore di soglia. Per quanto riguarda il numero di connessioni simultanee SN-SN, questo valore di soglia è sempre nell'intervallo 40-50. Per quanto riguarda le connessioni simultanee ON-SN, il valore di soglia è nel range 100-160 nel campus del Polytechnic e nel range 55-70 nella rete residenziale.

Siccome ogni giorno nella rete di KaZaA ci sono circa 3 milioni di peer, possiamo affermare che nella rete sono presenti 25000-40000 SN. Inoltre, ogni SN è connesso a circa lo 0,1% (50/4000) degli altri SN nella rete.

Lo studio [11] ha determinato una rete fortemente dinamica. Infatti, nonostante il numero di connessioni simultanee giunge ad un valore di soglia, le connessioni individuali cambiano frequentemente.

Utilizzando il KaZaA Sniffing Platform, sono state effettuate delle misurazioni sulla durata delle connessioni TCP ON-SN e SN-SN su un periodo di sette giorni.

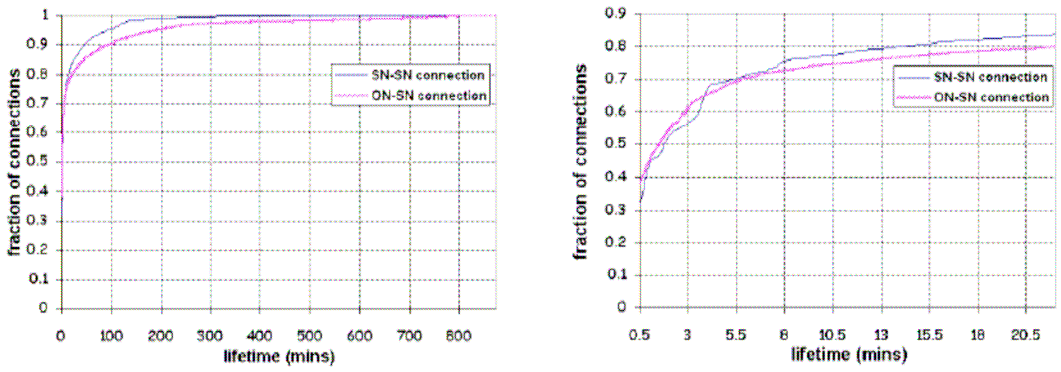


Figura 13: Distribuzione delle connessioni nel loro arco di vita

La Figura 13 mostra i risultati: sono state esaminate, su un periodo di 12 ore, un totale di 5206 connessioni ON e 3850 connessioni SN con il SN del dispositivo. Le durate medie delle connessioni ON-SN e delle connessioni SN-SN sono, rispettivamente, di 34 minuti e 11 minuti. Inoltre un notevole 38% delle connessioni ON-SN e un 32% delle connessioni SN-ON durano meno di 30 secondi.

Attribuiamo il largo numero delle brevi connessioni ON-SN a due fattori:

1) Un ON verifica se i SN candidati sono attivi inviando loro dei pacchetti UDP, successivamente il ON inizia le connessioni simultanee con i SN disponibili. Al termine il ON sceglie solo uno dei SN e si disconnette dagli altri. Per questo motivo il processo di instaurazione di connessioni ON-SN genera connessioni di breve durata
2) Ancora molti utenti utilizzano il client di KaZaA-Lite, il quale, come abbiamo già osservato, durante il processo di richiesta, salta da un SN all'altro utilizzando connessioni di brevissima durata

Il largo numero delle brevi connessioni SN-SN è stato attribuito a due fattori:

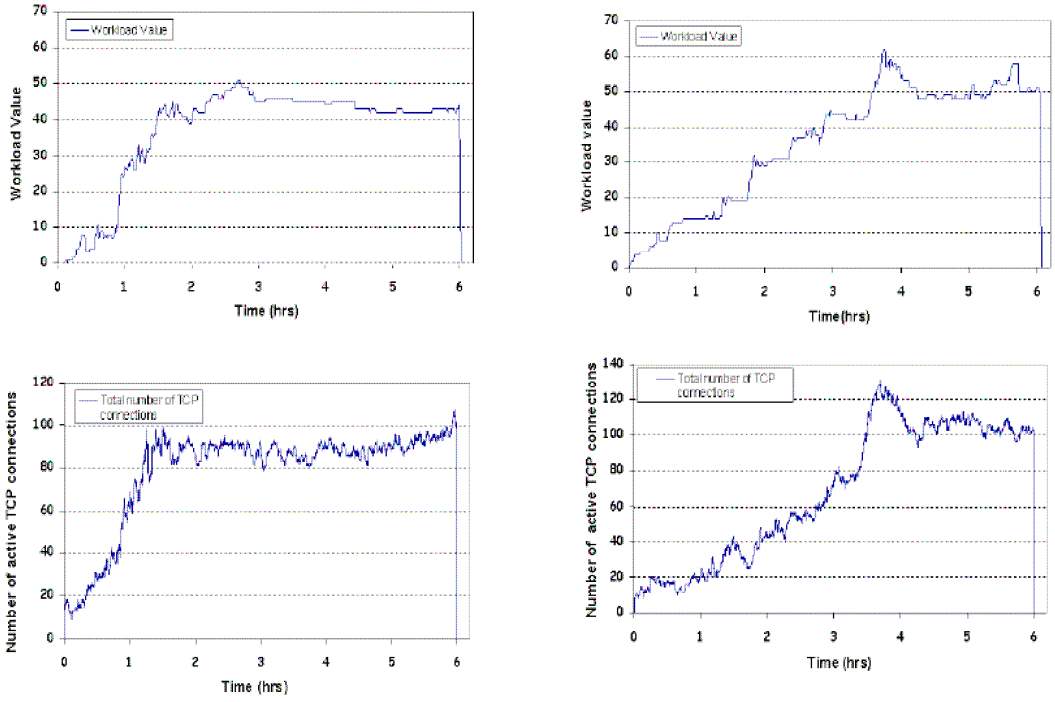
1) La ricerca dei SN di altri SN avviene generalmente con un piccolo Workload
2) Nella Overlay Network, i SN si connettono fra di loro solo con lo scopo di scambiarsi la SN refresh list.

1.6.5 Workload e Locality

Una caratteristica cruciale della rete ad due livelli è il criterio che un ON impiega per selezionare il SN padre. E' stato osservato [11] che, quando un ON stabilisce un collegamento con il SN padre, quest'ultimo fornisce al ON una lista di 200 SN. Inoltre questa lista contiene solo un sottoinsieme di tutti i SN che il padre mantiene nella sua cache. E' stato ritenuto, quindi, che i peer di KaZaA, nei collegamenti ON-ON e SN-SN, per la selezione del nodo vicino, seguano principalmente due criteri: Workload e Locality.

- **Workload:** ogni ON mantiene una SN refresh list nel Registro di Windows. Per ogni SN della lista sono presenti tre attributi: l'indirizzo IP, il numero di porta e il Workload. L'obiettivo ora è quello di capire se e come i valori di Workload influenzino la selezione del SN padre. Non si conosce l'esatta definizione di

Workload, ma, come si può notare dalla Figura 14, c'è una chiara correlazione tra il Workload stesso e il numero di connessioni che il SN sostiene.



(a) ON-SN connection establishment (b) SN-SN connection establishment

Figura 14: Legame fra il valore di Workload e il numero di connessioni TCP del SN

All'inizio il ON sceglie un sottoinsieme di SN, generalmente 5, dalla SN refresh list come candidati per diventare il suo SN padre. Un ON, quindi, tiene in grande considerazione il Workload del SN quando deve scegliere i candidati. Per verificare questa ipotesi, è stato utilizzato il KaZaA Sniffing Platform costringendo un ON del dispositivo a connettersi e, dopo poco tempo, a disconnettersi dalla rete Overlay. Ogni volta che il ON cerca di connettersi, esso sceglie un sottoinsieme dei SN dalla sua SN refresh list nel Registro di Windows. E' stato individuato il traffico dei segnali ed è stato determinato il sottoinsieme dei SN. Inoltre sono stati estratti i 200 SN dalla SN refresh list del ON. E' stato, poi, calcolato il valore medio dei Workload dei SN appartenenti al sottoinsieme scelto e il valore medio dei Workload dei SN presenti nella SN refresh list, ripetendo queste misurazioni ogni 30 minuti.

La Figura 15 visualizza il risultato: il client di KaZaA mostra una chiara preferenza per i SN con un basso valore di Workload.

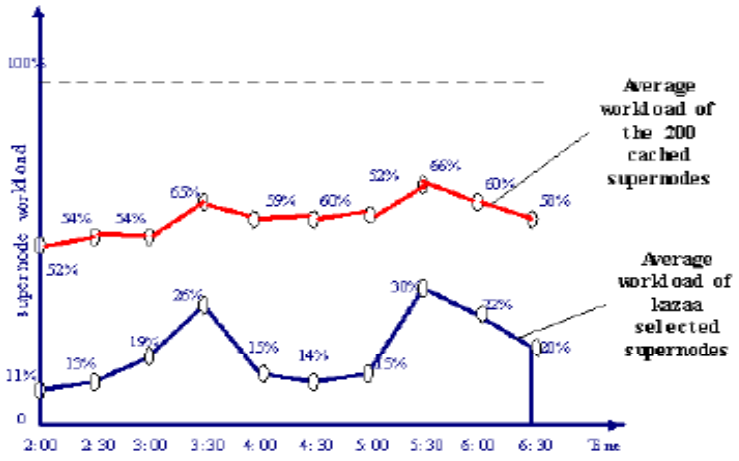


Figura 15: Preferenza per i SN con un basso Workload

- **Locality:** è stato ritenuto che tengano in grande considerazione la posizione, località, sia il ON, quando seleziona un SN padre, sia i SN quando selezionano il SN vicino nella Overlay Network. Per verificare tale ipotesi, sono stati effettuati due esperimenti.

  Nel primo esperimento [11] è stato utilizzato il Ping per calcolare il Round-Trip-Time (RTT) dal SN del KaZaA Sniffing Platform ai ON e SN, con i quali è connesso nella rete. La Figura 16 mostra la distribuzione di questi RTT: circa il 60% delle connessioni SN-SN hanno un RTT minore di 50 msec e circa il 40% delle connessioni ON-SN hanno un RTT minore di 5 msec. E' necessario valutare tali risultati sapendo che i segnali tra la costa est degli U.S.A. e l'Europa hanno un RTT di circa 180 msec, mentre quelli fra Nord America e Asia hanno un RTT di circa 180 msec.

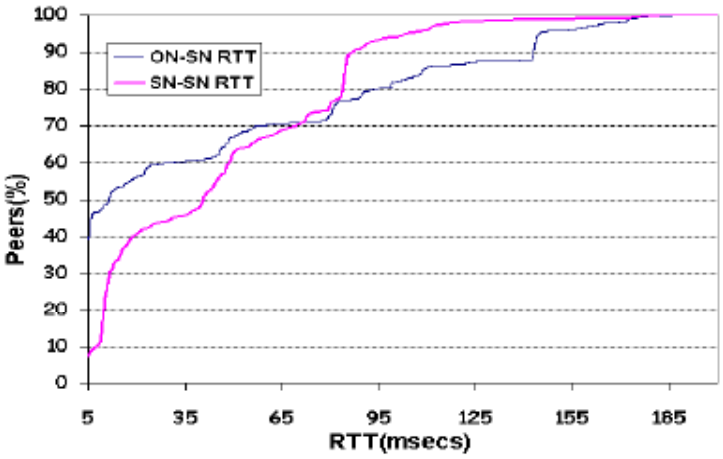
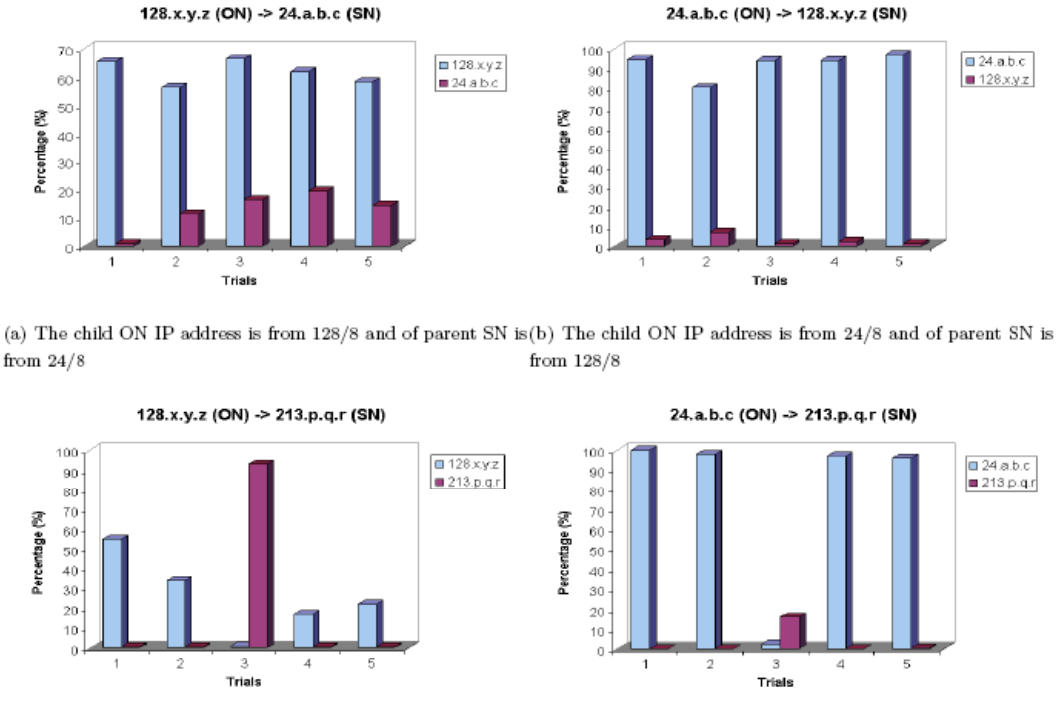


Figura 16: Misurazione del Round Trip Time

Il secondo esperimento [11] è focalizzato sui prefissi IP. E' stato utilizzato il KaZaA Probing Tool, che, come abbiamo descritto il precedenza, può connettersi con un SN specificato e ottenere la sua SN refresh list.



(a) The child ON IP address is from 128/8 and of parent SN is from 24/8

(b) The child ON IP address is from 24/8 and of parent SN is from 128/8

(c) The child ON IP address is from 128/8 and parent SNs are from 213/8

(d) The child ON IP address is from 24/8 and parent SNs are from 213/8

Figura 17: Posizione prefisso IP

In Figura 17 viene fornita la percentuale di SN, nella SN refresh list, che hanno un prefisso IP simile al SN padre e al ON figlio. In particolare, in Figura 10a è stato connesso un ON con indirizzo IP 128.x.y.z con un SN con indirizzo IP 24.a.b.c; è stato fatto l'opposto in Figura 10b connettendo un ON di IP 24.a.b.c con un SN di IP 128.x.y.z. Sia il ON figlio che il SN padre erano situati in U.S.A. In Figura 10c e 10d sono stati connessi due ON situati in U.S.A. di IP rispettivamente 128.x.y.z e 24.a.b.c con un SN padre di IP 213.p.q.r situato in Svezia. Dalle Figure 10a e 10b si può osservare come un'alta percentuale di SN abbia un prefisso IP simile al ON figlio; mentre questa percentuale scenda leggermente nella Figura 10c, perché i SN situati in Europa hanno una minore conoscenza dei SN situati negli U.S.A. Si può, quindi, concludere dicendo che i SN della refresh list hanno un prefisso IP che tende ad essere simile al prefisso IP del ON figlio. Perciò, quando un SN prepara la SN refresh list per un ON, il SN include, nella lista, i SN che nella rete sono vicini al ON figlio.

CAPITOLO 2
RICERCA DI FILE IN ARCHITETTURA P2P

2.1 Chord

Il sistema **Chord** [12] è un efficiente servizio di ricerca distribuito basato sul protocollo Chord e viene utilizzato per creare un'applicazione di file sharing peer-to-peer.

Tale sistema può essere confrontato con FreeNet: come quest'ultimo, Chord è decentralizzato, simmetrico, e si adatta automaticamente quando un host entra o esce dal sistema. A differenza di FreeNet, però, in Chord le richieste vanno sempre a buon fine oppure falliscono definitivamente.

Gli obiettivi nella realizzazione del sistema Chord sono:

- Scalabilità
- Disponibilità
- Bilanciamento delle operazioni
- Dinamismo
- Aggiornabilità

Il sistema Chord è costituito da due programmi: il client e il server. Il client è una libreria che prevede due importanti funzioni per l'applicazione di file-sharing: inserimento dei valori sotto una chiave e modifica dei valori di una data chiave. Il server implementa due interfacce: accettazione della richiesta proveniente da un client locale e comunicazione con gli altri server.

L'applicazione di file-sharing utilizza il sistema Chord per memorizzare i collegamenti fra i nomi dei file e l'indirizzo IP dei server che contengono i file. Chord mappa i nomi dei file in identificativi di chiave con una funzione crittografica hash (SHA-1). Il valore è un array di byte contenente una lista di indirizzi IP. Il client

e il server utilizzano programmi scritti in C++ e comunicano attraverso una connessione TCP.
Il protocollo Chord effettua essenzialmente una operazione: data una chiave, esso determinerà il nodo responsabile per la chiave. Gli elementi principali sono:

- **Chiave**: un identificativo di m bit ottenuto applicando una funzione hash ad una sequenza di byte
- **Valore**: un array di byte
- **NodeID**: un identificativo univoco di un nodo composto da m bit (spesso ottenuto applicando una funzione hash all'indirizzo IP del nodo)

Chord prevede 5 operazioni principali:

1) **insert(chiave,valore)**: inserimento di una coppia chiave-valore in r nodi distinti
2) **lookup(chiave)**: Chord trova la coppia chiave-valore da qualche nodo del sistema, e ritorna il valore al chiamante
3) **update(chiave,nuovo-valore)**: aggiornamento del valore relativo alla chiave richiesta. Tale operazione sopperisce alla mancanza di una operazione di delete
4) **join(n)**: ingresso di un nodo n nel sistema
5) **leave( )**: uscita di un nodo dal sistema

Dai realizzatori del sistema viene scelto un m abbastanza grande per abbassare la probabilità dei casi di collisione, quando, cioè, due nodi hanno lo stesso identificativo. Altrimenti si può aggiungere un suffisso univoco all'identificativo di ogni nodo per assicurare la univocità dell'identificativo del nodo. La possibilità di collisione negli identificativi delle chiavi non è considerata importante.
I nodi sono organizzati all'interno di un "anello logico" formato da chiavi di m bit. Le chiavi sono assegnate ai nodi in maniera semplice: ogni chiave, *k*, è assegnata al

primo nodo il cui identificativo, *id*, è uguale o segue *k* all'interno dell'anello. Questo nodo è chiamato *nodo successore* della chiave *k*, ed è definito come *successor(k)*. In altra parole, il *successor(k)* è il primo nodo che incontriamo muovendoci lungo l'anello in senso orario partendo da *k*.

Inoltre, è la stessa funzione hash a garantire una distribuzione uniforme delle chiavi e dei nodi all'interno dell'anello.

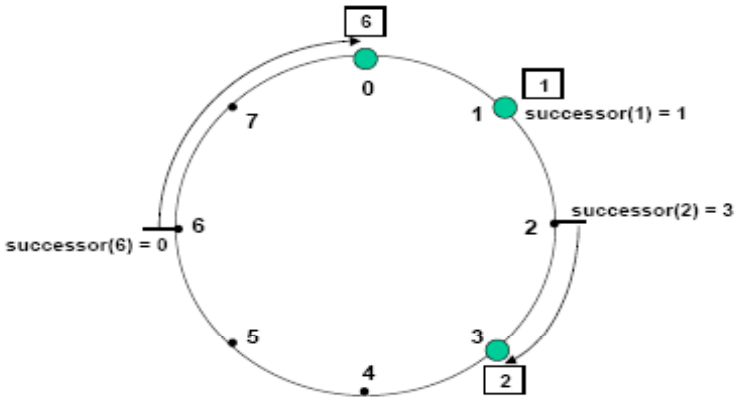


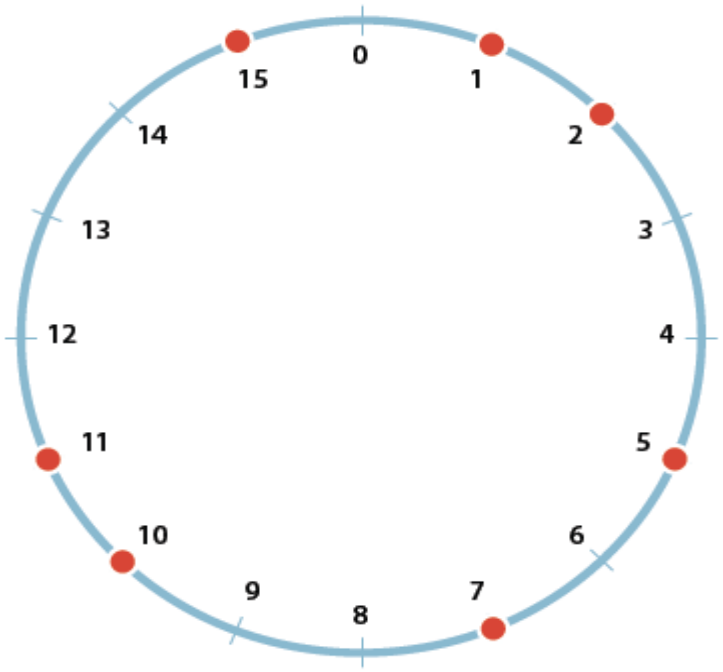
Figura 18: Esempio di rete Chord

La Figura 18 mostra l'esempio di una rete composta da 3 nodi 0, 1 e 3, i quali mantengono le chiavi 1, 2 e 6. m è uguale a 3, quindi gli identificativi sono di 3 bit. Siccome il successore della chiave 1, lungo i nodi nella rete, è il nodo 1, la chiave 1 è assegnata al nodo 1. Analogamente, il successore della chiave 2 è il nodo 3, perché esso è il primo nodo che troviamo muovendoci in senso orario lungo l'anello. Infine il successore della chiave 6 è il nodo 0.

Ogni nodo n possiede una tabella di routing con m righe chiamata *finger table*.

L'i-esima riga della tabella del nodo n punta al nodo restituito dalla funzione successor(n + 2^(i-1)). Dato un nodo n, nella sua tabella di routing, possiamo notare 4 colonne: la prima contiene i valori: n+2^0, n+2^1, n+2^2, n+2^3, …, n+2^m-1; la seconda contiene le rispettive chiavi; la terza contiene l'intervallo INT nel quale andremo a cercare il successore, la quanta contiene il successore stesso. Il successore è il primo nodo all'interno dell'intervallo considerato oppure il primo nodo dopo l'intervallo procedendo in senso orario lungo l'anello.

Esempio: una rete con i nodi 1, 2, 5, 7, 10, 11, 15 e relative finger table.



1	KEY	INT	SUCC.
1+2^0	2	{2,3}	2
1+2^1	3	{3,5}	5
1+2^2	5	{5,9}	5
1+2^3	9	{9,1}	10

2	KEY	INT	SUCC.
2+2^0	3	{3,4}	5
2+2^1	4	{4,6}	5
2+2^2	6	{6,10}	7
2+2^3	10	{10,2}	10

5	KEY	INT	SUCC.
5+2^0	6	{6,7}	7
5+2^1	7	{7,9}	7
5+2^2	9	{9,13}	10
5+2^3	13	{13,5}	15

7	KEY	INT	SUCC.
7+2^0	8	{8,9}	10
7+2^1	9	{9,11}	10
7+2^2	11	{11,15}	11
7+2^3	15	{15,7}	15

10	KEY	INT	SUCC.
10+2^0	11	{11,12}	11
10+2^1	12	{12,14}	15
10+2^2	14	{14,2}	15
10+2^3	2	{2,10}	2

11	KEY	INT	SUCC.
11+2^0	12	{12,13}	15
11+2^1	13	{13,15}	15
11+2^2	15	{15,3}	15
11+2^3	3	{3,11}	5

15	KEY	INT	SUCC.
15+2^0	0	{0,1}	1
15+2^1	1	{1,3}	1
15+2^2	3	{3,7}	5
15+2^3	7	{7,15}	7

E' importante notare che ogni nodo memorizza informazioni solamente riguardo un piccolo numero di altri nodi.

2.1.1 Lookup(k)

La finger table di un nodo n può non contenere abbastanza informazioni per determinare il successore di una chiave k arbitraria. Per risolvere ciò, viene utilizzata la funzione di *lookup(k)*: ogni nodo sceglie, attraverso la tabella di routing, il finger più lontano la cui chiave precede, o è equivalente, la chiave cercata, e inoltra la richiesta al nodo corrispondente. Questo metodo dà la certezza che, in un numero finito di passi, la richiesta raggiungerà il nodo responsabile della chiave cercata.

Esempio: ritornando alla rete precedente, supponiamo che il nodo 10 voglia conoscere in nodo responsabile della chiave 6. Il nodo 10 capisce, dalla sua finger table, che deve rivolgersi al nodo 2 ed effettua verso di esso la richiesta lookup(6). Il nodo 2, a questo punto, controlla nella sua tabella e nota che il responsabile della chiave 6 è il nodo 7, quindi, risponde al nodo 10.

2.1.2 Join(n) e Leave( )

In una rete dinamica i nodi possono entrare e uscire in ogni momento, ma, per permettere tutto ciò, devono essere rispettate due proprietà:

1) le finger table dei nodi devono essere corrette
2) ciascuna chiave k deve essere gestita dal nodo successor(k)

Quando un nuovo nodo n vuole entrare nella rete, viene utilizzata la funzione *join(n)*. Per rispettare le due proprietà appena citate, devono essere effettuate 3 operazioni:

A) Inizializzazione del predecessore e dei finger
B) Aggiornamento dei predecessori e dei finger di altri nodi nella rete
C) Trasferimento delle chiavi

A) Inizializzazione del predecessore e dei finger:
Calcolare la finger table è semplice: per ogni riga i delle tabella, il nuovo nodo n chiede ad un qualsiasi altro nodo, già nella rete, di calcolare successor(n+2^(i-1)).
Per calcolare il predecessore:

- n chiede ad un nodo qualsiasi di calcolare n'=successor(n)
- n chiede a n' chi è il suo predecessore
- il predecessore di n' diventa ora predecessore di n

B) Aggiornamento dei predecessori e dei finger di altri nodi nella rete:
Quando un nuovo nodo n entra nella rete, esso potrebbe diventare il finger e/o il successore di altri nodi delle rete. Ipotizziamo che, dopo l'ingresso di n nella rete, esso debba diventare l'i-esimo finger di un nodo p. Questo può accadere se e solo se:

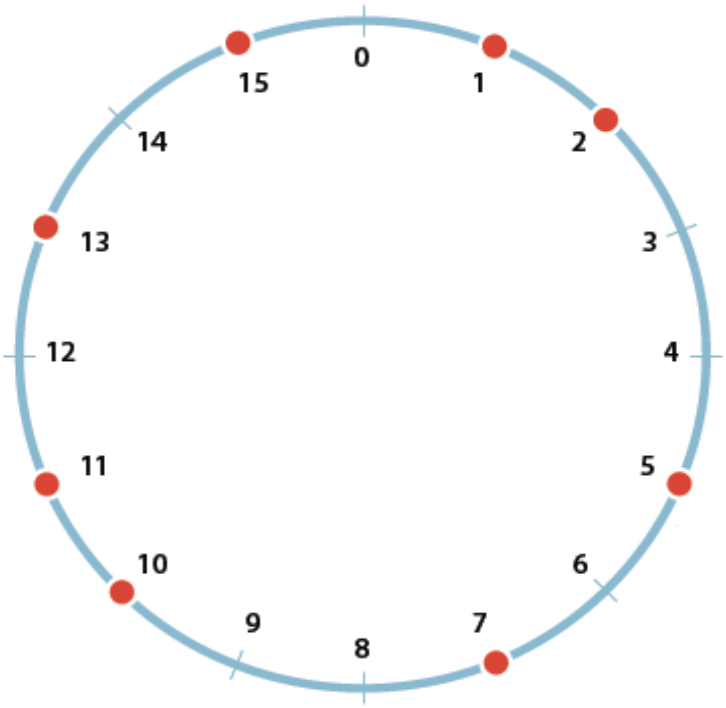
1) p precede n di almeno 2^(i-1) chiavi
2) il nodo m puntato dall'i-esimo finger di p soddisfa m=successor(n)

Il primo nodo, p, che può soddisfare entrambe le condizioni è l'immediato predecessore di n-2^(i-1). Per ogni finger i, l'algoritmo inizia a percorrere l'anello in senso antiorario partendo dalla chiave n-2^(i-1) aggiornando l'i-esimo finger di ogni nodo p' per il quale n=successor(p'+2^(i-1)).

C) Trasferimento delle chiavi:

Tutte le chiavi per cui n è diventato successore vengono spostate in n. Questa operazione può essere effettuata con una ispezione di tutte le chiavi gestite da successor(n).

Esempio: data la rete vista nell'esempio precedente con 7 nodi (1, 2, 5, 7, 10, 11, 15), facciamo entrare il nuovo nodo 13. Possiamo così notare i cambiamenti (in grassetto) in alcune finger table di alcuni nodi.



1	KEY	INT	SUCC.
1+2^0	2	{2,3}	2
1+2^1	3	{3,5}	5
1+2^2	5	{5,9}	5
1+2^3	9	{9,1}	10

2	KEY	INT	SUCC.
2+2^0	3	{3,4}	5
2+2^1	4	{4,6}	5
2+2^2	6	{6,10}	7
2+2^3	10	{10,2}	10

5	KEY	INT	SUCC.
5+2^0	6	{6,7}	7
5+2^1	7	{7,9}	7
5+2^2	9	{9,13}	10
5+2^3	**13**	**{13,5}**	**13**

7	KEY	INT	SUCC.
7+2^0	8	{8,9}	10
7+2^1	9	{9,11}	10
7+2^2	11	{11,15}	11
7+2^3	15	{15,7}	15

10	KEY	INT	SUCC.
10+2^0	11	{11,12}	11
10+2^1	**12**	**{12,14}**	**13**
10+2^2	14	{14,2}	15
10+2^3	2	{2,10}	2

11	KEY	INT	SUCC.
11+2^0	**12**	**{12,13}**	**13**
11+2^1	**13**	**{13,15}**	**13**
11+2^2	15	{15,3}	15
11+2^3	3	{3,11}	5

13	KEY	INT	SUCC.
13+2^0	**14**	**{14,15}**	**15**
13+2^1	**15**	**{15,1}**	**15**
13+2^2	**1**	**{1,5}**	**1**
13+2^3	**5**	**{5,13}**	**5**

15	KEY	INT	SUCC.
15+2^0	0	{0,1}	1
15+2^1	1	{1,3}	1
15+2^2	3	{3,7}	5
15+2^3	7	{7,15}	7

L'uscita di un nodo dalla rete viene effettuata tramite *leave()*. L'algoritmo usato da un nodo per lasciare la rete è basato sugli stessi concetti dell'ingresso, *join(n)*, del nodo nella rete.

Osservazione: in un sistema con N nodi e K chiavi:

1) E' alta la probabilità che un nodo sia responsabile di K/N chiavi
2) Ciascun nodo mantiene, nella finger table, informazioni relative ad altri O(logN) nodi
3) E' alta la probabilità che una ricerca, lookup(k), si concluda in O(logN) passi
4) E' alta la probabilità che, per completare l'operazione di entrata o di uscita di un nodo dalla rete, siano necessari O(log^2N) messaggi

2.1.3 Location table

Oltre alle informazioni sul routing, ottenute attraverso la finger table, il sistema Chord utilizza anche una *location table*, che contiene i nodi che Chord ha recentemente scoperto durante il funzionamento del protocollo. Essa è una memoria cache che mappa gli identificativi dei nodi con i loro indirizzi IP e porte. Gli identificativi dei nodi, che sono presenti nelle finger table, vengono inseriti nella location table. La tabella viene utilizzata per migliorare l'esecuzione dell'operazione di ricerca: invece di scegliere dalla finger table il nodo che è il più vicino predecessore delle chiave, il server Chord sceglie dalla location table il nodo che è un vicino predecessore e che è vicino nelle rete.

L'attuale implementazione utilizza una procedura di lookup iterativa, ma possiamo considerare di passare ad una procedura di lookup ricorsiva, per la quale le query viaggiano sempre nella direzione della loro destinazione finale. In questo modo, il protocollo fornisce tutti i nodi che sono stati visitati per risolvere la query e, quindi, vengono riempite correttamente le righe della location table.

Inoltre la location table viene adoperata per provare a risolvere il problema dei nodi falliti. Quando succede ciò, la procedura di ricerca sceglie un altro nodo dalla location table che è un vicino predecessore e dirige le query verso quel nodo. Il server Chord,

a questo punto, elimina il nodo fallito dalla sua location table e, se il nodo è presente anche nella sua finger table, aggiorna la tabella.

```
// ask node n to find id's successor
n.find_successor(id)
  n' = find_predecessor(id);
  return n'.successor;

// return closest finger preceding id
n.closest_preceding_finger(id)
  for i = m downto 1
    if (finger[i].node ∈ (n, id))
      return finger[i].node;
  return n;
```

```
// ask node n to find id's predecessor
n.find_predecessor(id)
  if (n == successor)
    return n; // n is the only node in network
  n' = n;
  while (id ∉ (n', n'.successor])
    n' = n'.closest_preceding_finger(id);
  return n';
```

Figura 19: Ricerca del nodo successore di un identificativo i*d*

```
// node n joins the network;
// n' is an arbitrary node in the network
n.join(n')
  if (n')
    init_finger_table(n');
    notify();
    s = successor; // get successor
    s.move_keys(n);
  else // no other node in the network to n itself
    for i = 1 to m
      finger[i].node = n;
    predecessor = successor = n;

// initialize finger table of local node;
// n' is an arbitrary node already in the network
n.init_finger_table(n')
  finger[1].node = n'.find_successor(finger[1].start);
  successor = finger[1].node;
  for i = 1 to m − 1
    if (finger[i + 1].start ∈ [n, finger[i].node))
      finger[i + 1].node = finger[i].node;
    else
      finger[i + 1].node =
          n'.find_successor(finger[i + 1].start);
```

```
// update finger tables of all nodes for
// which local node, n, has became their finger
n.notify()
  for i = 1 to m
    // find closest node p whose i^th finger can be n
    p = find_predecessor(n − 2^(i−1));
    p.update_finger_table(n, i);

// if s is i^th finger of n, update n's finger table with s
n.update_finger_table(s, i)
  if (s ∈ [n, finger[i].node))
    finger[i].node = s;
    p = predecessor; // get first node preceding n
    p.update_finger_table(s, i);

// if p is new successor of local stored
// key k, move k (and its value) to p
n.move_keys(p);
  for each key k stored locally
    if (p ∈ [d, n))
      move k to p;
```

Figura 20: Ingresso di un nodo nella rete

2.1.4 Chord Project

Il progetto Chord [13] è stato realizzato dal gruppo Parallel & Distributed Operative Systems (PDOS) (http://pdos.csail.mit.edu). Chord funziona su ogni sistema che supporta SFS, come Linux, FreeBSD, OpenBSD e Solaris. Smart File System (SFS) è un file system journaling, che, come tutti i file system di tipo journaled, una volta ricevuti, da un applicativo, i dati, che devono essere conservati nel sistema operativo, provvede, prima, a memorizzare le operazioni che deve compiere su un file di log; in seguito effettua la scrittura fisica dei dati sulla periferica di memoria di massa (ad esempio, il disco rigido); infine, registra nuovamente sul file di log, le operazioni che sono state effettuate. Esso è di libera distribuzione ed è scritto in C. E' stato ottimizzato per garantire scalabilità e integrità. Utilizza diverse grandezze di blocco a partire da 512 byte (2^9) fino a 32768 byte (2^15), e la grandezza massima totale della partizione è variabile, può essere 1 Terabyte (1024 Gigabyte) se applicato a blocchi di 512 byte, ma può ulteriormente aumentare, a seconda della grandezza massima di blocco adoperata. Una delle caratteristiche più importanti consiste nel raggruppare sia voci di directory multiple in un singolo blocco, sia blocchi di metadati insieme in cluster. Per tenere traccia dello spazio libero, viene utilizzata una bitmap. L'integrità dei metadati viene mantenuta da un log di rollback ("annullamento") di tutti i cambiamenti fatti ai metadati in un periodo di tempo determinato. Prima viene scritto il log sul disco, poi i blocchi di metadati vengono sovrascritti direttamente. Quindi, se il sistema dovesse andare in crush, la volta successiva che il file system verrà montato, quest'ultimo si accorgerà della procedura incompleta e ritornerà, con una procedura di rollback, all'ultimo stato coerente.

Dal sito sourceforge.net, bisogna scaricare una recente versione CVS di SFS, perché il codice-base di Chord fa uso di alcune funzioni che non sono disponibili nella versioni precedenti, come la 0.7.2. Per effettuare l'installazione, c'è bisogno di alcuni strumenti:

- GCC 2.95
- Autoconf, automake, GNU make

- Berkeley DB (scaricabile da http://www.spleepycat.com)

Il processo di installazione è documentato con precisione nella home page di SFS, ma possiamo riassumerlo in questi passaggi:

- lanciare *./setup* nella directory sorgente
- creare una directory d'installazione
- lanciare lo script *configure*, situato nella directory sorgente, dalla directory d'installazione
- lanciare *gmake* (o *make*)

Dovrebbe essere visualizzato:

```
%  cd ~/src/sfs1
%  ./setup
+ gm4 libsfs/Makefile.am.m4 > libsfs/Makefile.am
+ gm4 svc/Makefile.am.m4 > svc/Makefile.am
+ uvfs/setup
+ chmod +x setup
+ aclocal
+ autoheader
+ automake --add-missing
....
%  mkdir ~/sfs-build
%  cd ~/sfs-build
%  ~/src/sfs1/configure --with-dmalloc
creating cache ./config.cache
checking for a BSD compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets ${MAKE}... yes
checking for working aclocal... found
checking for working autoconf... found
checking for working automake... found
checking for working autoheader... found
```

```
...
% gmake
```

Il processo potrebbe durare diversi minuti e richiede almeno 700MB di spazio libero sul disco. Per ottenere il sorgente di Chord si può far uso di un CVS anonimo, per il quale non c'è bisogno di essere iscritti al gruppo PDOS.
Per verificare le sorgenti ottenute, bisogna lanciare i seguenti comandi:

```
% cvs -d :pserver:anoncvs@cvs.pdos.lcs.mit.edu:/cvs login
Logging in to :pserver:anoncvs@cvs.pdos.lcs.mit.edu:2401/cvs
CVS password: _press return_
% cvs -d :pserver:anoncvs@cvs.pdos.lcs.mit.edu:/cvs co -P sfsnet
```

Il processo di configurazione di Chord è basato su *automake* e, oltre a SFS, sono richiesti:

- Berkeley DB v4.x (scaricabile da http://www.spleepycat.com)
- Gtk

I comandi sono:

```
% cd src/sfsnet
% ./setup
+ gm4 svc/Makefile.am.m4 > svc/Makefile.am
+ chmod +x setup
+ aclocal
+ autoheader
+ automake --add-missing
...
% mkdir ~/chord-build
% cd ~/chord-build
% ~/src/sfsnet/configure --with-dmalloc --with-sfs=$HOME/sfs-build/
creating cache ./config.cache
checking for a BSD compatible install... /usr/bin/install -c
```

```
checking whether build environment is sane... yes
checking whether make sets ${MAKE}... yes
checking for working aclocal... found
checking for working autoconf... found
checking for working automake... found
checking for working autoheader... found
checking for working makeinfo... found
checking host system type... i386-unknown-freebsdelf4.3
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
% gmake
```

Avviare Chord è semplice, ma non completamente banale. Sono necessari, infatti, un database demone asincrono, *adbd*, un demone separato di sincronizzazione, *lsd* e un demone assistente, *syncd*.
Il database *adbd* viene lanciato in questo modo:

```
$ ../utils/adbd -h
./utils/adbd: invalid option -- h
Usage: adbd -d db -S sock
```

Quando viene lanciato senza opzioni, *adbd* memorizza il suo database in una directory */var/tmp/db* e inserisce una socket unix in */tmp/db-sock*.
Il secondo, *lsd*, prevede un collezione di opzioni. Inizialmente, conviene lanciarlo senza opzioni per essere sicuri che la sua configurazione sia andata a buon fine:

```
% ./lsd
Usage: lsd -j hostname:port -p port
[-d <dbsocket or dbprefix>]
[-v <number of vnodes>]
[-S <sock>]
[-C <ctlsock>]
[-l <locally bound IP>]
```

```
[-m [chord|debruijn]]
[-b <debruijn logbase>]
[-s <server select mode>]
[-L <warn/fatal/panic output file name>]
[-T <trace file name (aka new log)>]
[-O <config file>]
```

Le opzioni più interessanti sono:

-j hostname:port	viene definita la posizione del nodo corrente
-p port	il database si mette in ascolto sulla porta UDP e sulla porta TCP +1
-d dbsock	viene definito il prefisso per i database utilizzati per la memorizzazione di dati persistenti
-v nvnodes	creazione di nodi virtuali *nvnodes* in un singolo processo
-l ipaddress	il database si mette in ascolto su uno specifico indirizzo locale
-m mode	utilizzo della modalità di routing selezionata

Il demone *syncd* viene lanciato in questo modo:

```
$ ./syncd
Usage: syncd -j hostname:port
[-L logfilename]
[-v <number of vnodes>]
[-d <dbprefix>]
[-e <efrags>]
[-c <dfrags>]
```

Quando un nodo vuole entrare nella rete Chord, deve contattare un nodo ben conosciuto, che permetta il suo inserimento. Il nodo deve specificare una porta, evitando che il sistema Chord la scelga per lui, e il nodo di necessario per il suo inserimento. Il primo nodo che entra nella rete dovrà indicare se stesso come nodo utile per l'ingresso nella rete. Il seguente esempio mostra proprio questa situazione:

un host *sure.lcs.mit.edu* specifica un numero di porta, 10000, e se stesso come nodo per l'inserimento.

```
% ../utils/adbd &
% ./lsd -j sure.lcs.mit.edu:10000 -p 10000
Chord: running on 18.26.4.29:10000
init_ChordID: my address: 18.26.4.29.10000.0
 1004828619:637146 myID is caa42d5de473ac83e5be5cd96cdcaa6f7b85da56
lsd: insert: caa42d5de473ac83e5be5cd96cdcaa6f7b85da56
 1004828622:099189 stabilize:
caa42d5de473ac83e5be5cd96cdcaa6f7b85da56 stable!
                                                with estimate # nodes 1
```

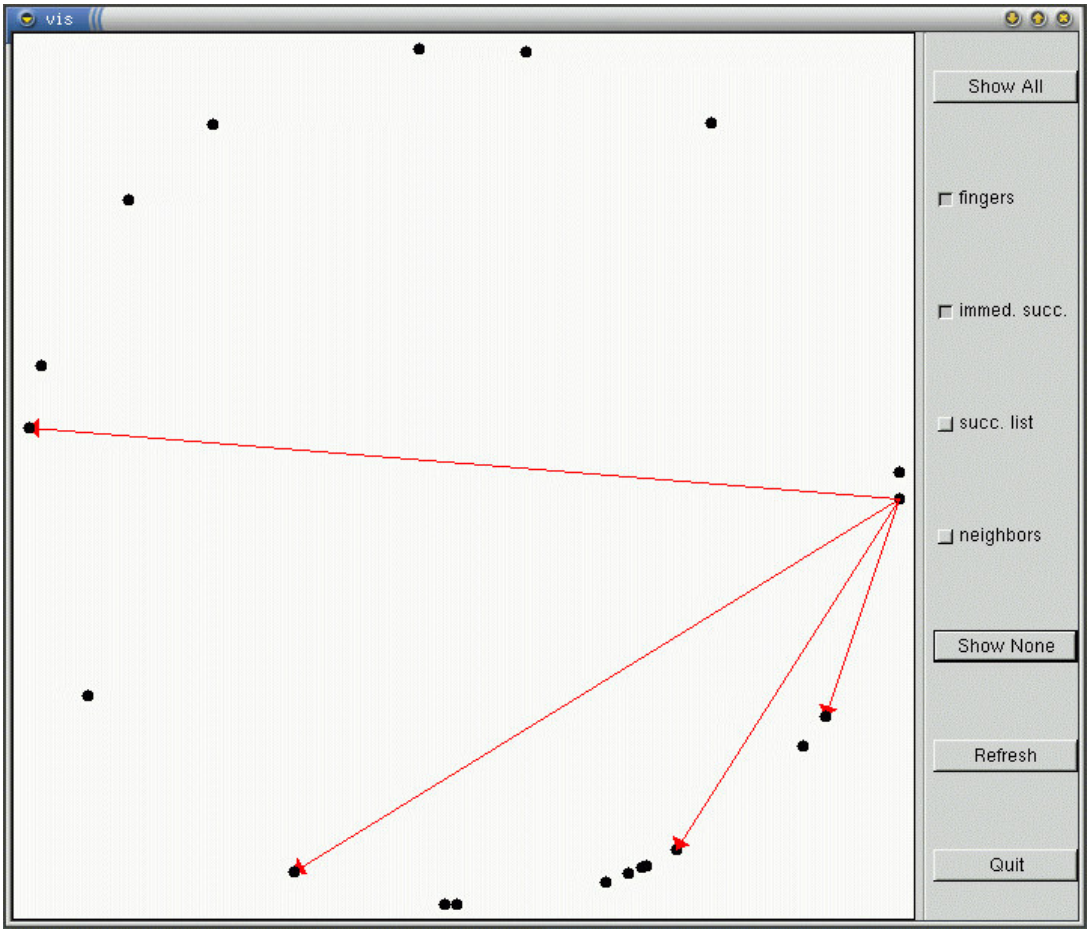
Per l'avvio di un nuovo nodo, non è necessario indicare il numero di porta, perchè *lsd* sceglierà una porta inutilizzata, ma è importante scrivere l'indirizzo del nodo dopo il parametro *-j*.

```
% ./lsd -j sure.lcs.mit.edu:10000
init_ChordID: my address: 18.26.4.29.2496.0
 1004829020:385471 myID is b678329a53d1a0a3e54fcd7a46d0d09d097fee34
lsd: insert: b678329a53d1a0a3e54fcd7a46d0d09d097fee34
 1004829027:570355 stabilize:
b678329a53d1a0a3e54fcd7a46d0d09d097fee34 stable!
                                               with estimate # nodes 12
```

A questo punto diventa fondamentale osservare lo stato dell'anello Chord per capire meglio le sue proprietà. Per rendere ciò più semplice, viene fornito un tool di visualizzazione, il *vis*, il quale richiede il Gtk. Se il Gtk è stato installato in una posizione non standard, bisogna specificare il prefisso utilizzando l'opzione di configurazione `--with-gtk-prefix`.
Il tool *vis* utilizza le stesse opzioni di *lsd* e, per farlo funzionare correttamente, bisogna puntarlo ad ogni nodo presente nel sistema. Ad esempio, nel seguente modo:

```
./vis -j sure.lcs.mit.edu:10000 [-a sec] [-f color file]
```

Dovrebbe, poi, comparire sullo schermo un'immagine del genere:



In questo esempio, vengono mostrati i finger di un singolo nodo. Per visualizzare le altre informazioni, bisogna premere il corrispondente bottone sulla destra. I bottoni *Show All* e *Show None* permettono, rispettivamente, di visualizzare tutti i nodi o nessuno. Il pulsante *Refresh* consente l'aggiornamento dell'immagine effettuando di nuovo la query ai nodi nel sistema.

CAPITOLO 3

ARCHITETTURE P2P BASATE SULL'ONTOLOGIA

3.1 HyperCuP

In una rete P2P la topologia è essenzialmente stabilita dai peer che sono in grado di comunicare fra loro: i nodi vicini sono quei nodi, ai quali, all'interno della rete, un peer può inviare dei messaggi. La rete P2P deve essere simmetrica: al suo interno ogni nodo deve avere le stesse capacità e gli stessi doveri. Il diametro di rete Δ, definito come il percorso più breve, in termini di hop, fra i nodi più distanti, deve essere un valore ragionevole, in quanto è una proprietà cruciale per quanto riguarda la ricerca e il broadcast. Il caso peggiore di Δ è O(n). E' importante, inoltre, che anche il traffico della rete sia distribuito uniformemente fra i nodi. Infine, la topologia deve fornire la ridondanza: la mancanza o il fallimento di un nodo non deve portare alla rottura del grafo o all'interruzione dei meccanismi di ricerca e di broadcast.

Essenzialmente, queste richieste affermano che ogni nodo dovrebbe essere in grado di diventare la root di uno spanning tree con tutti i nodi della rete. Si arriva così a realizzare una struttura efficiente, mostrata in Figura 21, con base b = 2, che, nella tridimensionalità, risulta essere un hypercube [14] con un numero b di nodi in ogni dimensione, una struttura che è stata già studiata nell'area delle macchine con più unità centrali, ma con assunzioni molto differenti.

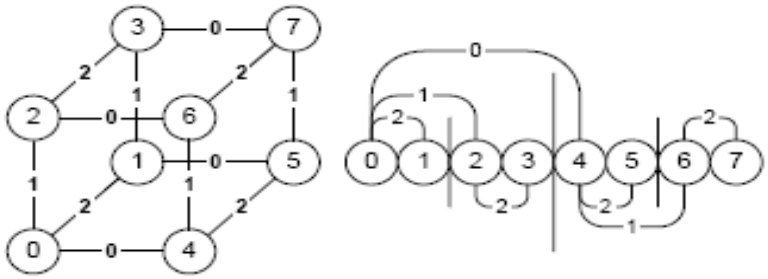


Figura 21: Struttura Hypercube

Un grafico hypercube completo consiste di N = b ^(Lmax+1) nodi ed è definita dal fatto che tutti i nodi hanno (b-1) * (Lmax + 1) vicini e (b – 1) in ogni dimensione, dove (Lmax + 1) è il numero delle dimensioni del cubo (in Figura 1, il cubo ha tre dimensioni con Lmax pari a 2). Il diametro di rete è $\Delta = \log_b N$.
Per descrivere meglio la topologia di un grafico G = (V,E), vengono utilizzate alcune definizioni. Per brevità, d'ora in poi considereremo sempre un hypercube con base binaria.

1. Gli spigoli sono contrassegnati: il nodo Y è definito come il i-vicino del nodo X o $Y = iN(X)$, il nodo Y è il vicino di X nella dimensione i. Per esempio, nella Figura 1, è il vicino del nodo 4 nella dimensione 2.
2. Gli spigoli sono indiretti: $\forall i(Y = iN(X) \leftrightarrow X = iN(Y))$, ad esempio, il nodo 4 è il vicino di 5 nella dimensione 2.
3. La dimensione di partenza è i = 0: il livello massimo di vicinanza è pari a Lmax. Ogni nodo X mantiene due insiemi che determinano la sua posizione nel grafico: un insieme di link dei nodi vicini $\Omega = \{\{\}, N1, N2, ..., Nn\}$ e un insieme associato di indirizzi dei nodi $A = \{localhost, addr1, addr2, ..., addrn\}$. Un nodo vicino è identificato da un link N e i messaggi possono essere inviati al suo indirizzo.
4. Un peer può avere solo un vicino per ogni livello di vicinanza.

E' stato definito uno schema broadcast, il quale garantisce che ogni nodo riceve il messaggio solo una volta. In questo modo esattamente N-1 messaggi sono richiesti per contattare tutti i nodi all'interno della topologia. Ogni peer può dare origine ad un processo di broadcast.
Ora spieghiamo il funzionamento dell'algoritmo attraverso un esempio, riferendoci alla topologia di Figura 21. Inizialmente il nodo 0 invia il suo messaggio di broadcast a tutti i suoi vicini, i nodi 4,2,1. Il nodo 4 riceve il messaggio attraverso il link di livello 0 e lo inoltra ai suoi due vicini di livello 1 e 2, i nodi 6 e 5. Nello stesso tempo, il nodo 2, che ha ricevuto il messaggio dal link di livello 1, lo inoltra al suo vicino di

livello 2, il nodo 3. Il nodo 6, invece, invia il messaggio al nodo 7, il suo vicino di livello 3.

In un hypercube la ricerca è fondamentalmente un broadcast con un TTL (Time To Live), un processo di broadcast in cui l'obiettivo è di far pervenire un messaggio ad un determinato peer in un predefinito numero massimo di hop.

3.1.1 Costruzione e mantenimento di una struttura Hypercube

In questo paragrafo, presenteremo l'algoritmo distribuito che permette ai nodi di costruire la topologia hypercube.

I principali obiettivi da raggiungere sono:

- Mantenere la simmetria della rete
- Consentire ad ogni nodo di accettare e integrare nuovi nodi
- L'entrata e l'uscita dei nodi devono utilizzare un numero ragionevole di messaggi senza superare un limite di traffico. Il nodo entrante, ad esempio, non deve comunicare il suo arrivo a tutti i nodi della rete.

Anche in questo caso, utilizzeremo un esempio per descrivere l'idea di base dell'algoritmo di costruzione e di mantenimento della rete: valuteremo cosa succede con l'ingresso di 9 peer e con l'uscita di un peer dalla rete.

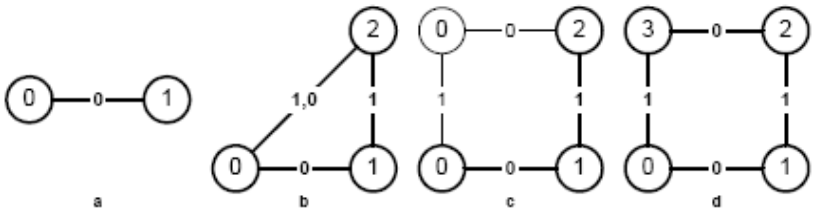


Figura 22: Costruzione della rete (1)

Fase iniziale: solo il nodo 0 è attivo.

Fase a: il peer 0 è contattato dal nodo 1, che vuole entrare nella rete P2P. Il peer 0 integra il peer 1 come vicino di livello 0, siccome attualmente non ha altri vicini. I due nodi stabiliscono un collegamento fra di loro, che viene contrassegnato da {0}. Un peer integra il nodo entrante nel suo primo livello di vicinanza libero.

Fase b: il peer 2 contatta uno dei due peer (nell'esempio, si assume che contatti il peer 1) per entrare nella rete. Il primo livello di vicinanza libero del peer 1 è il livello 1, siccome quest'ultimo possiede già il vicino di livello 0, il peer 0. A questo punto il peer 1 diventa il responsabile a fornire al peer 2 tutti i link necessari; esso infatti deve avere tutti i livelli di vicinanza correntemente esistenti, in modo tale che possa garantire il completamento di un eventuale processo di broadcast. Nel nostro esempio, siccome il peer 1 ha due vicini, ai livelli 0 e 1, anche il peer 2 dove avere due vicini agli stessi livelli. Quindi, il peer 1 diventa il vicino di livello 1 del peer 2 e, siccome non ci sono alternative, il peer 1 seleziona il peer 0 con vicino di livello 0 per il peer 2. Il peer 0 può essere solo un momentaneo vicino di livello 0 per il peer 2, perché esse possiede già un vicino di livello 0, il peer 1. Il peer 0 in questo momento copre una posizione vacante dell'hypercube.

Fase c: il peer 3 vuole entrare nella rete. Valuteremo tre casi. Il peer 3 contatta: 1) il peer 0, 2) il peer 1, 3) il peer 2.

1) Il peer 0 segue le regole generali integrando il peer 3 sul suo primo livello di vicinanza vacante, il livello 1. A questo punto il nuovo peer coprirà la posizione temporanea che il peer 0 manteneva nell'hypercube.
2) Il peer 1 integrerà il peer 3 sul livello di vicinanza 2. In questo modo, si apre una nuova dimensione per l'hypercube e si crea un momentaneo sbilanciamento con alcuni peer che mantengono più link di altri. Comunque la struttura si bilancerà da sola dopo un po' di tempo, perché l'informazione sulle posizioni vacanti si propaga su tutta la rete. In questo modo il nodo entrante contatterà un nodo che ha almeno una posizione vacante da riempire. Solo nei casi estremi sarà necessario un intervento attivo per riportare il bilanciamento nella struttura.

3) Il peer 2 decide di integrare il peer 3 come un nuovo, e non temporaneo, vicino di livello 0. Siccome il peer 0 attualmente copre la posizione che sarà occupata successivamente del peer 3, la responsabilità per la gestione dell'integrazione del nuovo nodo è affidata al peer 0. Di conseguenza, il peer 3 diventerà il nuovo vicino del peer 2 attraverso il peer 0. Per i peer della rete è sempre possibile contattare, con un solo hop, il nodo a cui hanno passato il controllo di integrazione. La situazione finale è mostrata dalla Figura 2d.

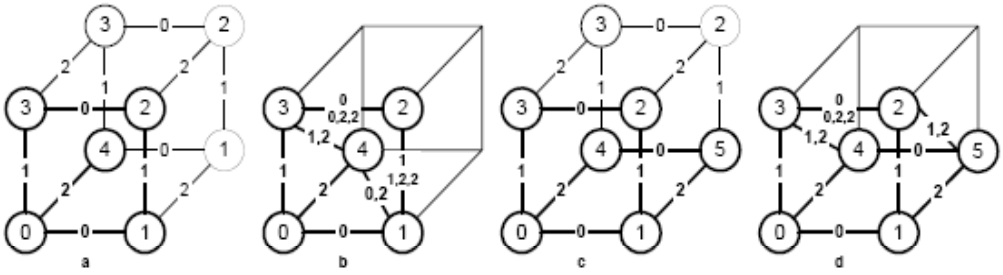


Figura 23: Costruzione della rete (2)

Fase d: il peer 4 contatta il peer 0. A questo punto, siccome un hypercube con due dimensioni non può ospitare 5 nodi, viene aperta una terza dimensione. Il peer 0 integra il peer 4 nel suo primo livello di vicinanza vacante, il nuovo nodo diventa il suo vicino di livello 2. Siccome ora le dimensioni sono tre, il peer 4 ha bisogno di tre vicini, uno per ogni livello di vicinanza. Inoltre, per quanto riguarda il peer 0, né il suo vicino di livello 0, il peer 1, né il suo vicino di livello 1, il peer 3, sono collegati con i loro vicini di livello 2, che essi dovrebbero associare al peer 4 come nuovi vicini. Quindi, il peer 1 diventa il vicino temporaneo di livello 0, mentre il peer 3 diventa il vicino temporaneo di livello 1, come si può vedere dalla Figura 3b.

Fase e: il peer 1 viene contattato per integrare il peer 5. Il peer 1 è attualmente mancante del vicino di livello 2, per questo motivo il peer 5 verrà integrato in questa posizione (Figura 3c). Il peer 5 diventa, inoltre, il vicino di livello 0 del peer 4, ma, anche per il nuovo peer, diventa necessario un collegamento temporaneo: il peer 5 diventa il vicino di livello 2 del peer 2 (Figura 3d).

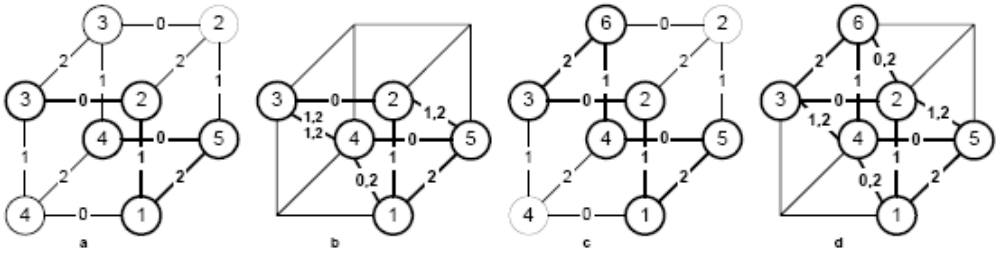


Figura 24: Costruzione della rete (3)

Fase f: assumiamo ora che,improvvisamente, il peer 0 abbandoni la rete. Il peer 0 lascia una posizione vacante, la sua originale posizione nel grafo. E' necessario, quindi, che un nodo, che ricopre posizioni multiple, trovi i successori per ogni sua posizione coperta nel grafo. Il nodo in questione è il peer 4, che stabilisce link temporanei per i precedenti vicini del peer 0. il peer 1 e il peer 3. La Figura 4a mostra, in seguito all'uscita del peer 0, la nuova distribuzione delle responsabilità di copertura; la Figura 4b descrive l'evoluzione della struttura.

Fase g: il peer 4 è contattato dal peer 6 e decide di integrarlo come suo nuovo vicino di livello1. Questa posizione è correntemente coperta dal peer3, perciò il peer 4 inoltra il controllo di integrazione al peer 3, come descritto dalla Figura 4c. In aggiunta il peer 3 integra il peer 6 come suo vicino di livello 2, giungendo così alla situazione mostrata dalla Figura 4d con la presenza di nuovi link temporanei.

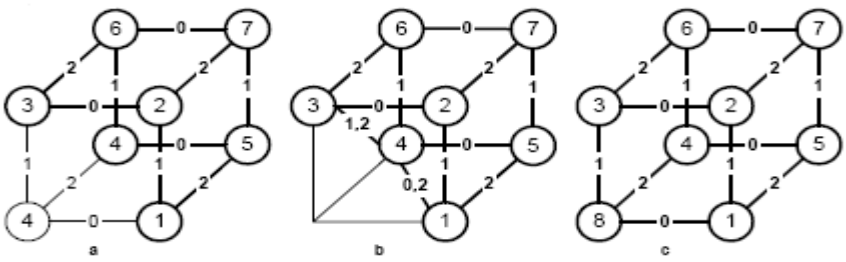


Figura 25: Costruzione della rete (4)

Fase h: il peer 6 viene contattato dal peer 7, il quale viene integrato come vicino di livello 0. Come mostrano la Figura 5a e la Figura 5b, quasi tutte le posizioni

dell'hypercube a tre dimensioni sono mantenute da peer attivi, solo il peer 4 copre due posizioni.

Fase i: il peer 8 contatta il peer 4, il quale integra il nuovo peer come vicino di livello 2. L'hypercube è, dunque, completo in ogni sua posizione.

3.1.2 Utilizzo di un routing basato sull'ontologia

Una rete P2P Hypercup rivela sia una buona scalabilità, sia bassi tempi di ricerca. Le prestazioni della rete possono essere comunque migliorate. Spesso le informazioni che i peer sono in grado di fornire possono essere organizzate in categorie per esprimere un concetto generale. I concetti possono essere raccolti in una ontologia globale, che definisce le relazioni fra i concetti esistenti. Nel caso in questione, non è stato usato solo un hypercube per l'intera rete, ma è stato realizzato un hypercube per ogni concetto, consentendo di raggiungere tutti i peer, i quali forniscono, in maniera efficiente, informazioni e servizi per questo concetto. Inoltre, l'organizzazione interna dei nodi di un hypercube è organizzata in modo tale che la rete possa supportare combinazioni logiche dei concetti dell'ontologia attraverso le query.

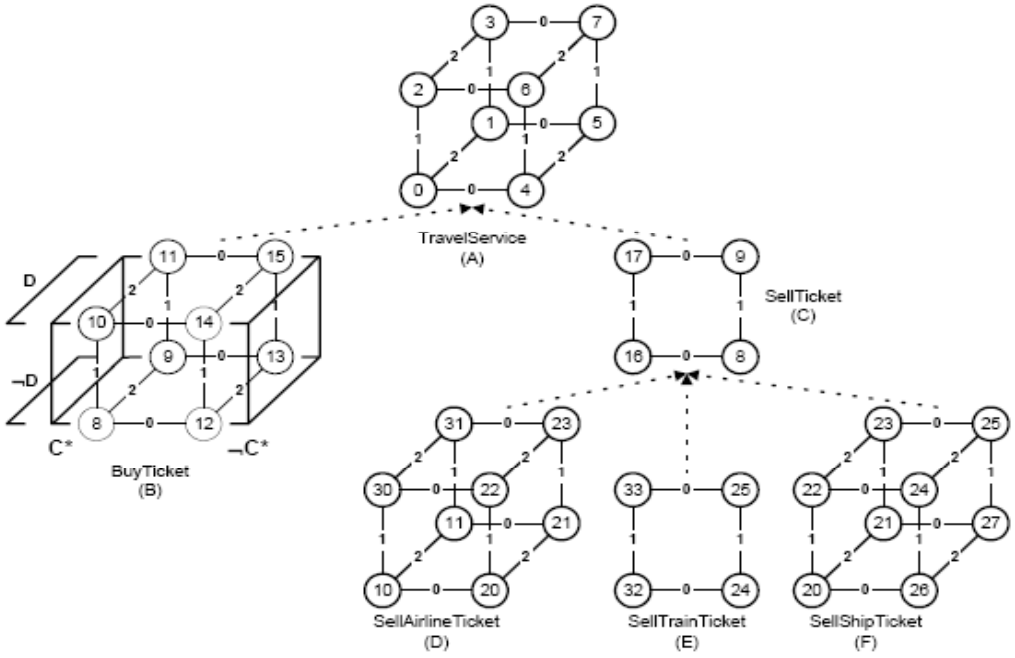


Figura 26: Topologia della rete basata sull'ontologia

Come descritto dalla Figura 26, per ogni concetto è stato costruito esattamente un hypercube, che contiene precisamente quei peer, che supportano direttamente il concetto stesso. E' importante notare che un nodo che è collegato a diversi concetti presenti nell'ontologia è membro di ogni hypercube che è associato ad uno di quei concetti. La rete consiste di sei hypercube, uno per ogni concetto ontologico, collegati fra di loro. In particolare, fra due hypercube connessi, il numero dei link è uguale al numero dei nodi dell'hypercube più piccolo, e i link sono ugualmente e deterministicamente distribuiti fra i nodi dell'hypercube più grande.

In ogni query viene specificato il concetto richiesto, in questo modo il peer tenta di guidare la query verso l'hypercube corretto e associato al concetto stesso. Il routing viene compiuto attraverso i link presenti fra gli hypercube. Inoltre, per evitare problemi di traffico, gli hypercube disposti sullo stesso livello, ad esempio, gli hypercube B e C, sono collegati fra loro.

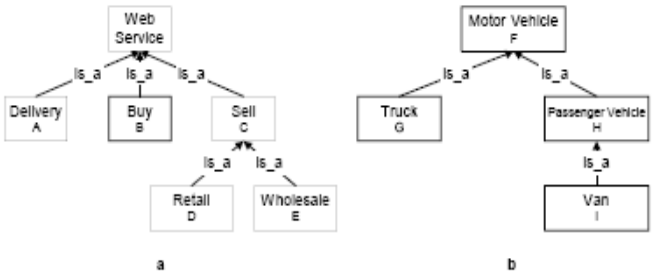


Figura 27: Due esempi

La Figura 27 mostra altri due esempi.

Il primo è un servizio Web descritto utilizzando varie ontologie in parallelo. Possono così essere inseriti hypercube che riguardano i venditori o di chi gestisce servizi di consegna, come mostrato dalla Figura 27a.

La Figura 27b, invece, mostra come un dominio di automobili possa essere descritto classificando le varie tipologie di macchine. Un venditore di auto, che vuole fornire un servizio Web, vorrebbe descrivere se stesso attraverso la combinazione di un

concetto dell'ontologia del servizio Web con alcuni concetti dell'ontologia del domino delle automobili, per esempio, $C \wedge G \wedge I$.

Il servizio di rete P2P deve fare in modo che la query venga inviata esattamente a quei peer che possono potenzialmente rispondere ad essa. Così, ad esempio, una query $B \wedge I \wedge \neg G$ deve essere inviata a quei nodi che comprano macchine, ma che non sono interessati ai camion. Per questo motivo è stato introdotto il concetto di cluster all'interno della topologia di rete hypercube. I peer, quindi, con interessi o servizi identici o simili sono raggruppati in cluster, i quali sono assegnati ad una specifica logica combinazione di concetti ontologici. In questo modo viene consentito di effettuare il routing verso gli adatti concetti cluster all'interno della topologia.

L'hypercube è, quindi, costituito da diversi cluster. Inoltre, ogni cluster contiene più di un unico peer, di conseguenza diventa necessario utilizzare delle coordinate, che definiscano la posizione del peer all'interno del cluster per poter poi propagare correttamente una eventuale query.

Il meccanismo di routing è diviso in due fasi: nella prima, la query viene inviata verso quei concetti cluster che contengono peer, ai quali è indirizzata. Nella seconda, viene effettuato il broadcast all'interno di ogni cluster scelto precedentemente, inoltrando ottimamente la query verso tutti i peer.

Come abbiamo visto precedentemente, i peer possono entrare all'interno dell'hypercube contattando ogni peer già presente nella rete. In particolare, il peer entrante definisce le sue capacità in termini di concetti contenuti, e deve essere integrato nel concetto cluster corrispondente alla sua descrizione. Se un peer descrive se stesso con concetti che non sono presenti nella rete, esso viene integrato nel concetto cluster più generale.

3.1.3 Implementazione di HyperCuP

Questa implementazione di **HyperCuP** [17] è stata realizzata da Pawel Bugalski e Slawomir Grzonkovski. E' scritta in java e offre funzioni come: creazione di nuove reti, connessione a peer aggiuntivi. Quando un nuovo peer si connette alla rete, esso deve conoscere solo un singolo indirizzo di uno dei nodi già presenti nella rete.

Questa connessione iniziale non ha alcuna influenza sulla posizione dei peer nella topologia finale della rete. Inoltre, è possibile inviare messaggi in ogni dimensione, dove per "dimensione" si intende un numero di peer vicini. Ciascun peer è connesso a un numero di nodi equivalente alla dimensione dell'hypercube. Durante l'uso di HyperCuP, il programmatore può facilmente inserire nel messaggio qualsiasi oggetto java attraverso l'implementazione dell'interfaccia Content. Se c'è un insufficiente numero di nodi nella rete (minore di quello richiesto dalla grandezza dell'hypercube), viene eseguita l'operazione di clonazione: un nuovo peer viene inserito in una dimensione già esistente e, attraverso quest'ultima, si connette ad uno dei peer presenti e, nelle altre dimensioni, con i cloni degli altri peer. Ogni nuovo peer entrante sostituisce uno dei cloni, finché non ci sono più nodi e diventa necessario aprire una nuova dimensione.

Sebbene, in teoria, l'implementazione esistente è abbastanza completa, essa soffre della mancanza di alcune funzioni necessarie per il lavoro giornaliero della rete P2P nell'ambiente reale. Prima di tutto, la rete deve essere in grado di gestire l'uscita dei peer, sia esplicita che implicita (ad esempio, come risultato di un crush o in seguito alla rottura di una connessione), e di riparare questa sua situazione. La gestione della compattezza della rete è basata su messaggi KEEP_ALIVE. L'attuale implementazione si focalizza sulla creazione della topologia della rete, ma non trae profitto da essa, perché non esiste ancora una implementazione dei messaggi generici. HyperCuP dovrebbe essere estesa per fornire quei messaggi, simili ai precedenti, necessari per il trattamento dei messaggi di ricerca.

Riassumendo, le funzioni già implementate sono:

- creazione di nuove reti
- connessione a peer aggiuntivi
- comunicazione con ogni indirizzo
- invio dei messaggi ai vicini e routing del messaggio
- invio di oggetti all'interno dei messaggi
- clonazione dei nodi

Le funzioni da realizzare sono:

- esplicita uscita dei peer
- implicita uscita dei peer
- riparazione dello stato della rete
- utilizzo di messaggi generici

I realizzatori di HyperCuP hanno pensato di utilizzare i servizi Web per la comunicazione fra i peer all'interno della loro implementazione piuttosto che altre soluzioni. In generale, i servizi Web sono leggermente più lenti, ma possono essere utilizzate molte piattaforme indipendenti, ben documentate e standardizzate. Un altro aspetto positivo di questa tecnologia è l'abilità di utilizzare diversi protocolli di trasporto come HTTP, SMTP o HTTPS. Allo scopo di eseguire questa operazione, Pawel e Slawomir stanno scrivendo un soap wrapper e saranno in grado di raggiungere questo obiettivo senza cambiare nulla nel codice di basso livello già scritto.
I servizi Web utilizzano il protocollo SOAP per le transazioni. I dati vengono trasferiti fra client e server come un semplice XML, quindi, teoricamente, chiunque intercetti il messaggio può leggerlo e cambiarlo. I servizi Web forniscono sicurezza principalmente attraverso il livello di trasporto. Attualmente, gli sviluppatori possono utilizzare uno di questi tre metodi:

- BASIC - autenticazione browser
- FORM – una form HTML scritta dall'utente
- CLIENT-CERT – certificato

E' possibile rendere sicuri i servizi Web a livello SOAP e di messaggio. Fortunatamente, questo livello è stato ampiamente soggetto di ricerca nei gruppi come OASIS (http://www.oasis-open.org/specs). Nel Marzo 2004, quest'ultimo ha terminato il lavoro e ha pubblicato una descrizione finale. Ora gli sviluppatori

possono utilizzare firme digitali, certificati, domini di fiducia e metodi di crittografia a livello di documento XML. Quindi, anche riguardo i servizi Web, è stata sviluppata e standardizzata una descrizione generale riguardo la sicurezza.

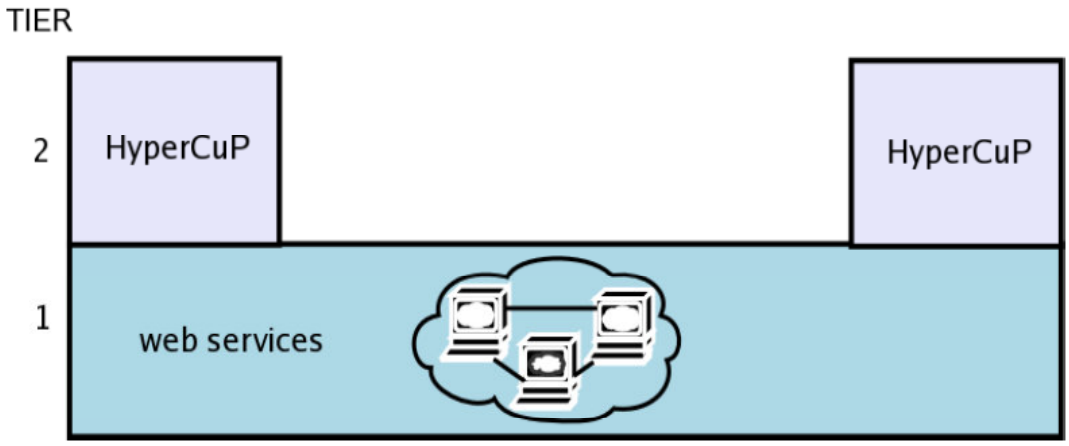


Figura 28: Architettura HyperCuP a due livelli

Allo scopo di avere un migliore conoscenza della futura applicazione HyperCuP, è stata realizzata una architettura a due livelli. Il primo livello deve fornire un'affidabile comunicazione fra due istanze di HyperCuP: esso è, infatti, il livello di trasporto. Questa operazione verrà eseguita da un wrapper speciale. Attualmente è in corso la implementazione del wrapper, come servizio Web su un server Tomcat dotato di un'applicazione Axis. Il secondo livello è l'attuale implementazione di HyperCuP, per la quale tutte le funzioni menzionate precedentemente devono essere realizzate.
Durante la scrittura del wrapper dovrebbero essere compiute due operazioni. Innanzitutto, lo stesso servizio Web deve essere creato. Esso deve "coprire" l'implementazione dell'interfaccia del peer. Ogni volta che un metodo, offerto dal servizio Web, sarà invocato, un'appropriata classe di messaggio dovrà essere trasmessa all'implementazione del peer attraverso un metodo socket del peer stesso. In secondo luogo, il metodo di invio della implementazione del peer deve essere messo da parte allo scopo di inviare messaggi di HyperCuP attraverso il livello del servizio Web. Questo codice dovrebbe invocare i metodi del servizio Web offerti dagli altri peer.

La figura seguente mostra lo Use Case diagram di HyperCuP:

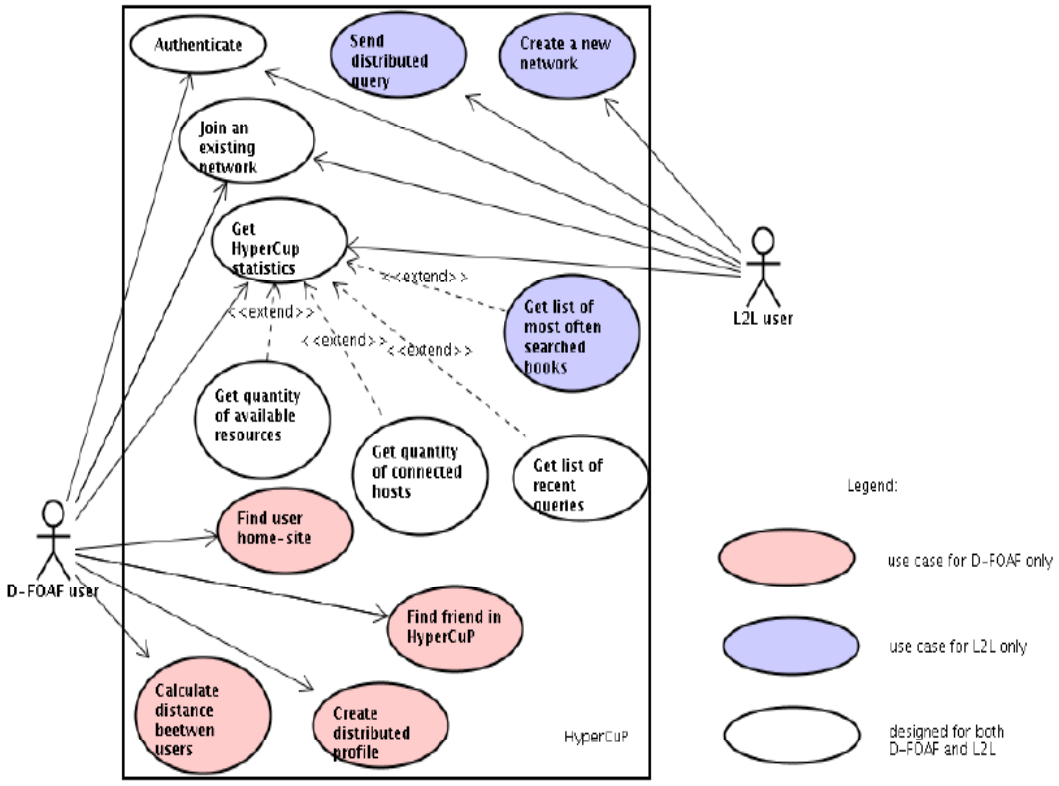


Figura 29: Use Case Diagram di HyperCuP

Analisi dell'Use Case L2L:

Di seguito, elenco alcuni Use Case per la Elvis Digital Library, che descriverò successivamente e che dovrebbe usare le funzionalità di HyperCuP per:

- la creazione di nuove reti di librerie
- gestire l'ingresso in un rete esistente
- ottenere le statistiche dell'hypercube (quantità di nodi connessi, quantità di libri disponibili, una lista dei libri più ricercati, una lista delle query recenti)
- l'invio di una query ad ogni libreria nella rete e raccogliere un'unica risposta
- l'autenticazione e l'autorizzazione di una libreria divenuta connessa alla rete

Analisi dell'Use Case D-FOAF:

Allo scopo di utilizzare effettivamente la topologia hypercube nel progetto FOAF-Realm (http://foafrealm.org), devono essere realizzate le seguenti funzionalità:

- ingresso in una rete esistente
- ricerca di un amico nell'hypercube
- ricerca della home-site dell'utente
- creazione di un profilo collettivo

Inoltre, dovrebbero essere considerate alcune funzioni, in particolare:

- ottenere le statistiche dell'hypercube (quantità di nodi connessi, quantità di risorse disponibili, una lista delle query recenti)
- calcolo della distanza fra due utenti
- autenticazione e autorizzazione durante la connessione al regno (realm)

3.1.4 Elvis Digital Library

La pubblicazione di documenti, come libri, testi scientifici e immagini, richiede una modalità per una facile gestione, per una ricerca efficiente e per una corretta visualizzazione. Queste richieste non possono essere fornite da una semplice pagina Web o da un server FTP. Una libreria virtuale deve provare a seguire la mente del lettore per presentare a lui l'informazione esatta che cerca, evitando ogni possibile perdita di tempo. E' importante, inoltre, progettare un modo per memorizzare tutti i documenti e renderli velocemente disponibili a tutti i potenziali lettori. Le modalità tradizionali per la pubblicazione dei testi non forniscono un controllo su chi accede alla risorsa, sulla stampa o sulla copiatura del testo, né uno strumento per il pagamento di queste attività.

La **Elvis Digital Library** [19] presenta la stessa architettura client-server di una comune applicazione Web. L'applicazione lato client è costituita da una pagina Web, scritta usando la tecnologia JSP (Java Server Pages), e da alcune applet. Queste sono

le responsabili principali della finale redazione dei dati richiesti e della presentazione al lettore. JSP è una tecnologia in qualche modo simile al PHP, ma più semplice da integrare con le prime applicazioni Java, come le applet, ed è più potente. L'applicazione lato server consiste di un insieme di servlet Java, che lavorano con i database XML e SQL. Queste servlet sono responsabili dell'estrazione dei dati conformi alla query ricevuta, dell'esecuzione della query e dell'invio di un'appropriata risposta in formato XML al client.

L'architettura è a tre livelli. Il livello più basso è responsabile della cooperazione con i database (invio delle query, raccolta delle risposte) e con il file system. Il livello centrale elabora le risposte provenienti dal livello più basso, portando i risultati al livello superiore, e, nell'altra direzione, elabora le richieste dal livello superiore, inviandole al livello inferiore. Il livello più alto è responsabile della presentazione dei dati in una modalità standardizzata, supporta le richieste dell'utente, ad esempio, le query di ricerca, mostra vari messaggi. In un'architettura di questo tipo è molto semplice inserire nuovi componenti, come un nuovo database o un nuovo strumento di ricerca, senza ricostruire l'intere applicazione. Tutto ciò migliora la scalabilità del sistema.

Uno dei più importanti vantaggi di Elvis è la conformità con il Semantic Web. Quest'ultimo è un'idea della "seconda generazione" di Internet, in cui le applicazioni Web possono trattare i dati che elaborano in maniera simile agli umani. Per questo motivo tutti i dati devono essere descritti utilizzando descrizioni standardizzate, chiamate "descrizioni semantiche", che possono identificare un oggetto nel mondo reale e possono presentare le sue proprietà e le sue connessioni con altri oggetti. Sicuramente è molto difficile (o impossibile), in una singola applicazione, descrivere il mondo intero e tutte le classi oggetto che possono esistere in esso, ma in questo contesto non è necessario. E' possibile, tuttavia, creare la descrizione di una piccola parte del mondo, un dominio. Questa descrizione, chiamata ontologia, è realizzata nella libreria Elvis. Tale ontologia prevede oggetti come "libro", "autore" e le loro proprietà, e memorizza queste informazioni con ogni oggetto presente nel database. In questo modo Elvis fornisce agli utenti una opzione per la "ricerca semantica", che

utilizza le descrizioni semantiche durante la esecuzione di una ricerca. Le ricerche semantiche hanno dimostrato di essere più efficaci di quelle classiche.

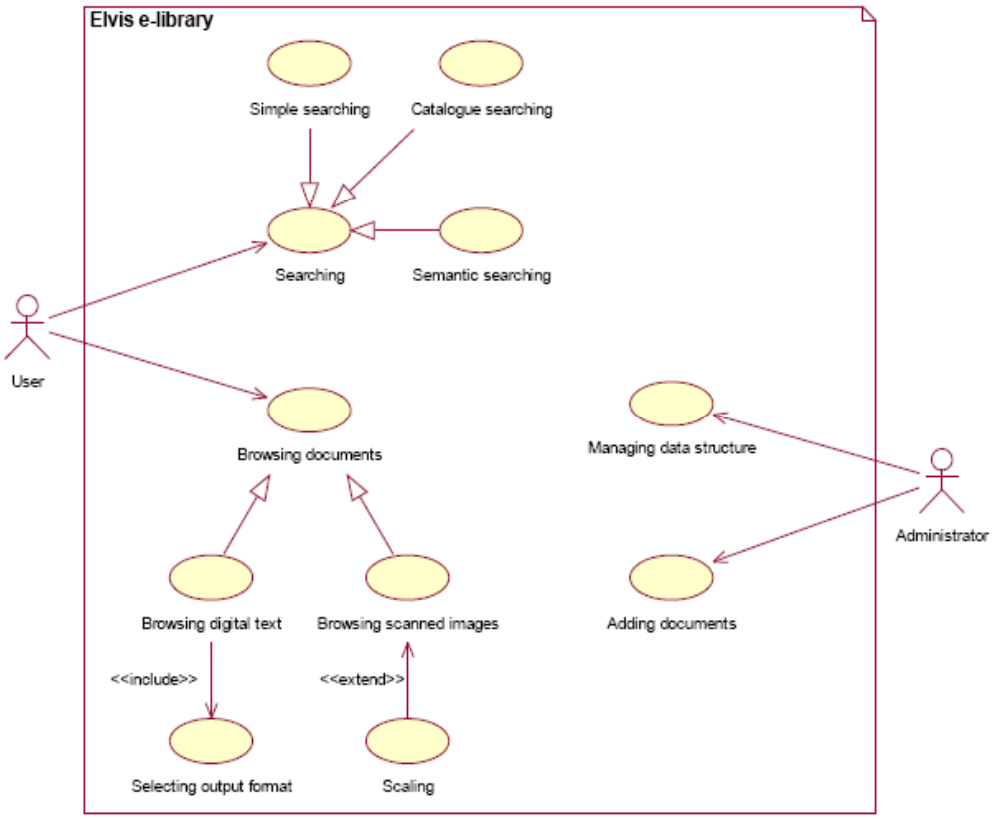


Figura 30: Use Case Diagram di Elvis Digital Library

Le funzionalità principali della libreria Elvis sono:

A) Memorizzazione dei dati

B) Manipolazione dei dati

C) Presentazione dei dati

A) Memorizzazione dei dati:

Sono tre i principali tipi di informazione, che vengono memorizzati in Elvis:

- Contenuti del documento: includono tutti i tipi di libri o riviste, che possono presentarsi sottoforma di testo digitale, di insieme di immagini o di altri altri file binari, come le presentazioni di Macromedia Flash
- Descrizioni del documento: forniscono informazioni riguardo il libro stesso e il suo autore. Esse sono conformi allo standard Dublin Core e utilizzano una ontologia per rendere più efficace il processo di ricerca
- Descrizioni della struttura del documento: forniscono informazioni riguardo i capitoli, i titoli e le pagine. Vengono utilizzate per facilitare la lettura e la visualizzazione dei documenti.

Una libreria virtuale dovrebbe fornire una modalità standard per memorizzare e mostrare i dati. Ci sono numerosi formati, i file .DOC, RTF, PDF, HTML e tanti altri. Alcuni di essi, poi, per essere letti, richiedono un software commerciale, un semplice browser Web, infatti, non è sufficiente. Per risolvere questo problema, Elvis memorizza i dai nel formato XSL:FO, utilizzando eXist, un database XML e Open Source. Il formato XSL:FO è basato sul formato XML, che consente alcune funzioni avanzate. Il principale vantaggio di XSL:FO è che ogni file XSL:FO è un file XML e, per questo motivo, può essere facilmente riformattato in HTML o PDF utilizzando un'appropriata trasformazione XSLT. Molti browser Web, come Internet Explorer o Mozilla, gestiscono la coppia XML+XSLT con facilità, mantenendo questa operazione completamente trasparente all'utente.

Il formato XSL:FO, tuttavia, non è gestito dai comuni editori di testo. Per importare i file, quindi, Elvis utilizza la libreria RTF2FO. Quest'ultima è l'unica parte commerciale della libreria Elvis. E' una libreria Java, che consente la conversione dei file RTF in file XML o XSL:FO. E' stata effettuata questa scelta perché i file RTF sono gestiti dai più utilizzati editori di testo, come Microsoft Word o OpenOffice.org Writer. La libreria RTF2FO è un prodotto Novosoft Inc. (http://www.rtf2fo.com).

Sfortunatamente, non ogni contenuto può essere memorizzato in un file XML, come , ad esempio, il contenuto binario. Per memorizzare i file binari, come le immagini JPEG e PNG, Elvis utilizza il file system UNIX/Linux. Ad esempio, una libreria

RTF2FO, durante l'importazione di un file RTF con delle immagini presenti al suo interno, estrarrà queste immagini e le memorizzerà sul disco, mentre il contenuto di testo verrà scritto nel file XML e memorizzato nel database eXist.

Per rendere più efficace la ricerca, ad ogni documento memorizzato viene associata una descrizione, che contiene informazioni, come autore, titolo, categoria e keywords.

Per raggiungere gli obiettivi di Elvis, è stata realizzata una ontologia, ElvisOnt, che consente di creare descrizioni semantiche utilizzando il linguaggio RDF. Essa è un insieme di regole che descrivono una piccola parte del mondo, in questo caso, gli oggetti come Libro, Autore o Keyword, le loro proprietà e le loro associazioni. Queste regole possono essere scritte attraverso il linguaggio RDF-S (RDF Schema).

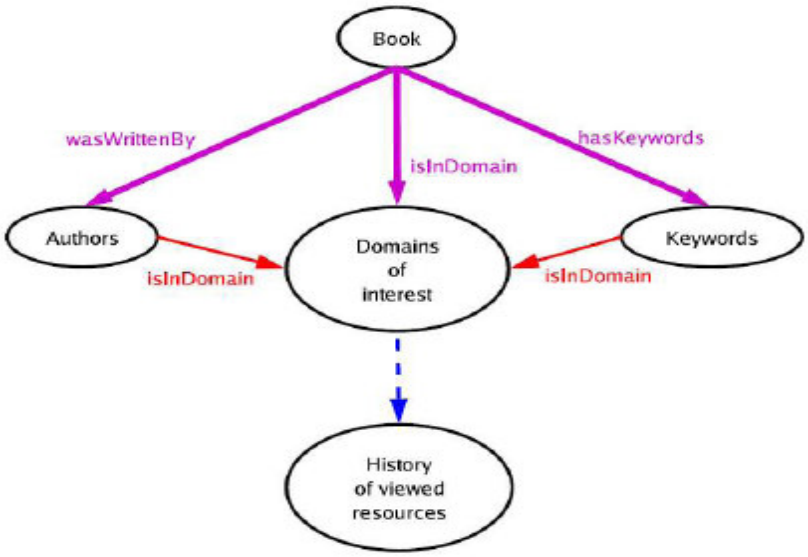


Figura 31: Una parte di ElvisOnt

Le descrizioni sono compatibili con il più comune formato di descrizione bibliografica, il Dublin Core, che consiste di 12 proprietà necessarie per definire un documento: autore, titolo e altri.

Il Dublin Core, ha, tuttavia, molti aspetti negativi: le sue proprietà non definiscono nessuna connessione semantica con altri concetti; il numero dei parametri è limitato,

solo 12, ed essi possono contenere solo una limitata quantità di informazione riguardo il documento che descrivono.
Per questi motivi, Elvis consente di memorizzare i dati addizionali nel formato MARC21. Anch'esso è uno standard per la descrizione delle risorse bibliografiche, ma è più esauriente. E' possibile, inoltre, convertire le descrizioni dal formato MARC21 al Dublin Core, ma comporta una perdita di informazione. Elvis, quindi, consente di aggiungere le descrizioni in MARC21 a quelle scritte in Dublin Core. Queste informazioni aggiuntive verranno utilizzate durante il processo di ricerca, incrementando le possibilità di trovare i documenti desiderati. Per velocizzare la loro elaborazione, le descrizioni in MARC21 vengono memorizzate nel formato MARC-XML, che è semplicemente una versione XML del MARC21. Naturalmente, è possibile passare dal MARC-XML al MARC21 utilizzando una trasformazione XSLT.
Elvis utilizza un sistema di descrizioni della struttura del documento, per il quale vengono memorizzati:

- titolo, autore, riassunto
- copertina
- capitoli
- parti multimediali
- vincoli di sicurezza (diritti di copiatura e stampa)

Questo sistema comporta diversi vantaggi, fra i quali, ad esempio, la possibilità di saltare da un capitolo ad un altro. Queste descrizioni vengono memorizzate nel database eXist in formato XML.
La ricerca full-text consiste nel confrontare la query con l'intero contenuto del documento. Questo metodo sarebbe estremamente lento se, ogni volta che viene inviata una query, dovesse essere analizzato tutto il contenuto. Per mantenere i vantaggi della ricerca full-text, Elvis utilizza un sistema a indici. La libreria Lucene è responsabile del posizionamento di ogni documento aggiunto al database. Il suo

contenuto viene analizzato e le parole molto comuni o insignificanti vengono eliminate, è il processo "taglio". Le parole risultano private dei comuni prefissi e suffissi e memorizzate in un indice full-text.

Elvis utilizza, inoltre, un sistema cookie per memorizzare i dati riguardanti un utente. I cookie sono piccoli file, memorizzati nel computer dell'utente, che contengono informazioni utili per individuare l'utente stesso, mantenendo la sua anonimità, e le sue preferenze. In questo modo, il sistema riesce a capire meglio le richieste dell'utente, ad esempio, durante le query di ricerca, e a rispondere ad esse.

B) Manipolazione dei dati:

Una caratteristica importante della libreria Elvis è la sua capacità di ricerca del documento. Gli altri strumenti di ricerca utilizzano algoritmi abbastanza semplici per scorrere il documento, confrontare l'informazione ottenuta con la query, e decidere se la risorsa è adeguata o no a quest'ultima. Successivamente mostrano i risultati, nei quali ci possono essere molte "hits" (se la query era troppo generale) o nessuna (se era troppo specifica). Inoltre, molti documenti trovati non rispondono a ciò che si stava cercando. Elvis utilizza nuovi metodi per assicurare che i risultati della ricerca rispondano alle aspettative. E' stato, quindi, implementato un algoritmo di ricerca con capacità estese.

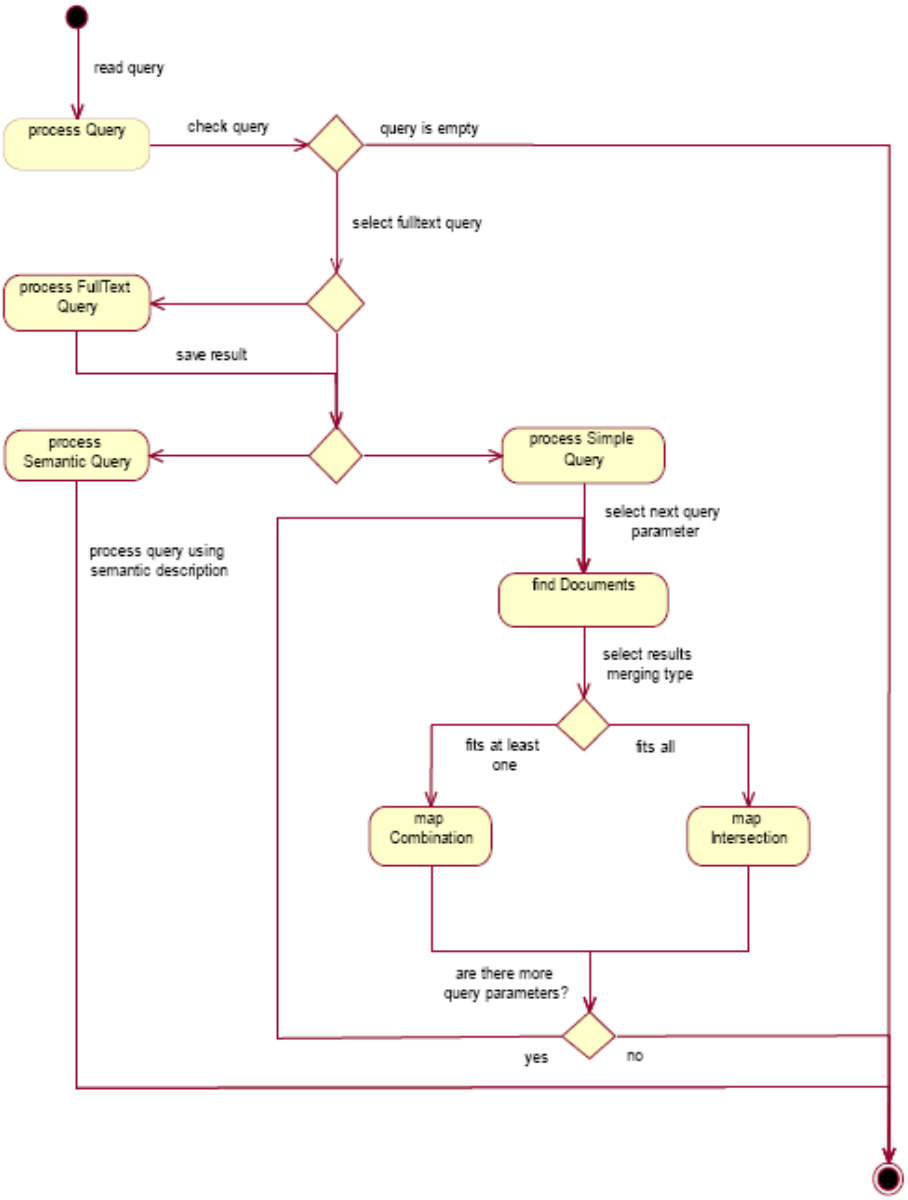


Figura 32: Algoritmo di ricerca

I 3 metodi di ricerca forniti sono:

1) **ricerca semplice**: è il metodo tradizionale. Elvis utilizza l'indice full-text della libreria Lucene e i principali campi di descrizione (autore, titolo, riassunto, parole chieve e dominio di interesse) per effettuare la ricerca delle parole della query. I risultati sono riuniti e mostrati all'utente senza effettuare operazioni aggiuntive.

2) **ricerca a catalogo**: l'utente può fornire liste separate di parole in campi differenti come full-test, titolo e autore. I risultati possono essere riuniti o intersecati.
3) **ricerca semantica**: dalla prospettiva dell'utente, è simile alla ricerca a catalogo. Infatti, l'utente riempie alcuni campi, fornendo le informazioni della query, ma l'algoritmo tenta di adeguare il numero dei risultati fra 5 e 25. Per fare ciò, esso trae vantaggio sia dalle connessioni semantiche fra i concetti, sia dall'informazione sui domini di interesse dell'utente memorizzata nei file cookie.

   Se i risultati sono troppo pochi (meno di 5), l'algoritmo prova a generalizzare la query. Ad esempio, se la ricerca veniva effettuata su un certo sotto-dominio, l'algoritmo prova ad effettuarla su un intero dominio. Questo processo viene chiamato "resolve general". Se i risultati sono ancora troppo pochi, l'algoritmo potrebbe provare a ricercare attraverso i sinonimi delle parole della query (questa funzione è in corso di sviluppo).

   Se i risultati sono troppi (più di 25), la query verrà modificata diventando più specifica in base alle preferenze dell'utente. Ad esempio, una ricerca viene effettuata sul dominio "costruzione". Elvis riconosce che l'utente molto spesso predilige i libri riguardanti la costruzione di ponti e, a questo punto, modifica la ricerca effettuandola su "costruzione di ponti". Questo processo viene chiamato "resolve nuances".

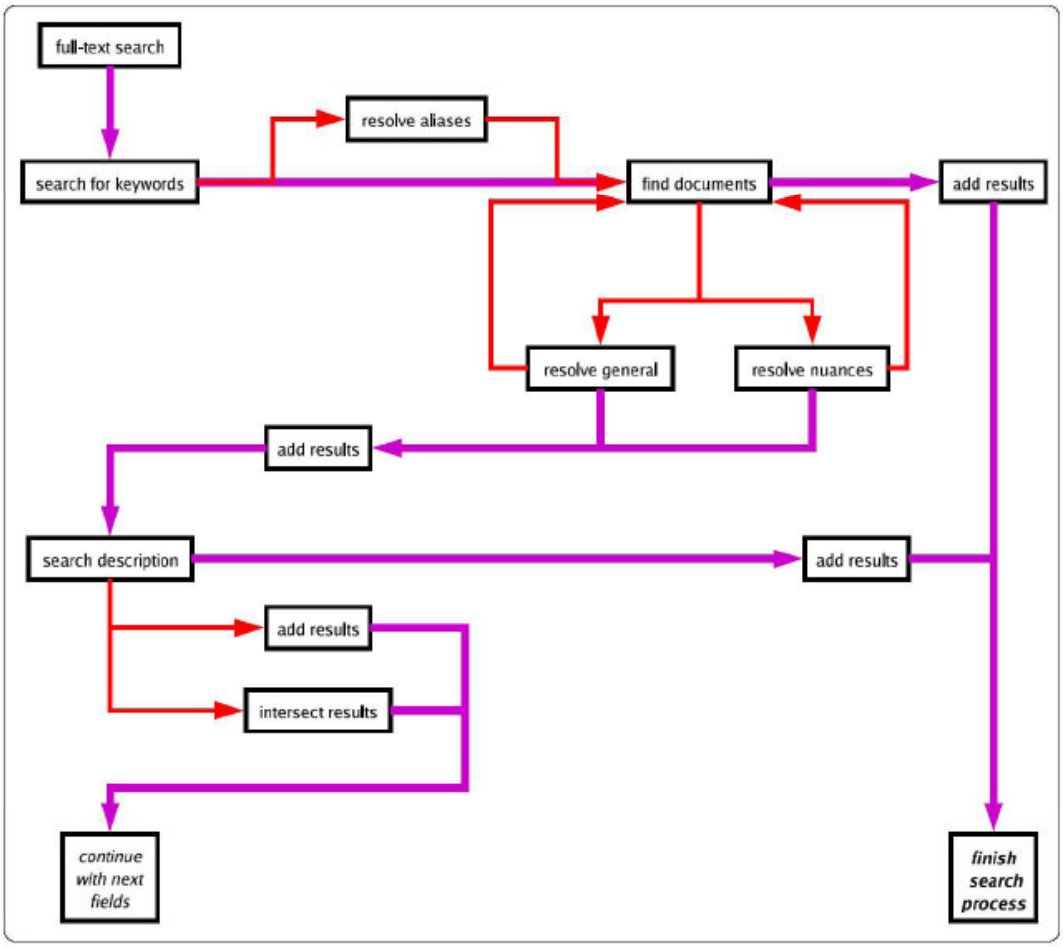


Figura 33: Processo di ricerca semantica

Per valutare la prestazione della ricerca semantica, sono state definite alcune metriche:

$$precision = \frac{f_t}{f_t + f_f} \cdot 100\% \qquad recall = \frac{f_t}{f_t + n_t} \cdot 100\% \qquad waste = \frac{f_f}{f_f + n_f} \cdot 100\%$$

dove:

f_t è il numero di risorse trovate che rispondono alle aspettative

f_f è il numero di risorse trovate che non rispondono alle aspettative

n_t è il numero di risorse che non sono state trovate, ma che rispondono alle aspettative

n_f è il numero di risorse che non sono state trovate e non rispondono alle aspettative

I risultati dei test sono mostrati nella tabella seguente, dalla quale si può notare che la ricerca semantica offre risultati migliori rispetto ai tradizionali strumenti di ricerca.

	typical		***semantic***		***ideal***
	avg	dev	avg	dev	avg
precision	53.3%	9.8%	89.9%	6.4%	100.0%
recall	54.0%	9.3%	89.9%	5.9%	100.0%
waste	7.3%	1.8%	2.9%	1.2%	0.0%

Elvis memorizza i vincoli di sicurezza considerandoli come informazioni riguardanti il documento. Essi consistono di:

- diritto di accesso al documento
- diritto di stampa del documento
- diritto di copiatura del documento (utilizzando il meccanismo di copia & incolla)

Quindi, se, ad esempio, il secondo e il terzo vincolo vengono settati come globali, nessuno può stampare o copiare il documento. Questo è molto importante nel caso in cui il libro in considerazione sia protetto dai diritti di autore, ma si voglia consentire la lettura ad ognuno.
Facendo uso della libreria FOAF-Realm (http://foafrealm.sf.net), gli sviluppatori stanno progettando di estendere il controllo di accesso con vincoli di sicurezza personalizzati, consentendo così ad alcuni utenti di effettuare determinate azioni su

determinati documenti. Inoltre, tutto questo risulta utile come parte del sistema dei micro-pagamenti. Questa è una nuova caratteristica della libreria Elvis, che obbliga gli utenti a pagare piccole somme di denaro per accedere, stampare e/o copiare una risorsa. Col termine "denaro", si può intendere sia soldi reali, sia differenti tipi di soldi virtuali, come i crediti. Quando un utente vuole accedere ad una risorsa non gratuita, questi verrà informato dell'obbligo di pagamento e potrà decidere se confermare l'operazione oppure no.

C) Presentazione dei dati:

Per l'utente, la parte più importante della libreria digitale è la sua interfaccia e le sue capacità; dove essere facile da usare e deve mostrare le informazioni in maniera chiara.

Attualmente, la libreria Elvis è stata realizzata per la Gdansk University of Technology Main Library, sotto il nome di WBSS (Wirtualna Biblioteka Sieci Semantycznej – Semantic Web Virtual Library). La sua pagina Web è disponibile all'indirizzo: http://www.wbss.pg.gda.pl.

Durante la lettura delle pubblicazioni disponibili, un utente può scegliere il formato di presentazione dei dati. I testi digitali possono essere automaticamente convertiti in HTML, PDF o RDF attraverso una appropriata trasformazione XSLT. Il formato HTML è il più compatibile ed è visualizzato da ogni Web browser, PDF e RDF sono i formati più utilizzati per la pubblicazione di documenti digitali in Internet.

Un'altra funzionalità utile di Elvis è la regolazione dell'immagine. Se la pubblicazione, che l'utente sta leggendo, presenta delle immagini JPEG al suo interno, una applet gli consente di effettuare alcune operazioni come lo zoom o la visualizzazione di solo una parte dell'immagine stessa.

Il tool di amministrazione di Elvis, JElvisAdmin, è un'applicazione Java stand-alone, che comunica con le principali applicazioni di Elvis (attraverso un'appropriata servlet) utilizzando il Java RMI (Remote Method Invocation). Quest'ultimo è un meccanismo simile a Unix RPC (Remote Procedure Call), che consente di chiamare le funzioni su un sistema remoto capace di elaborare le chiamate RMI.

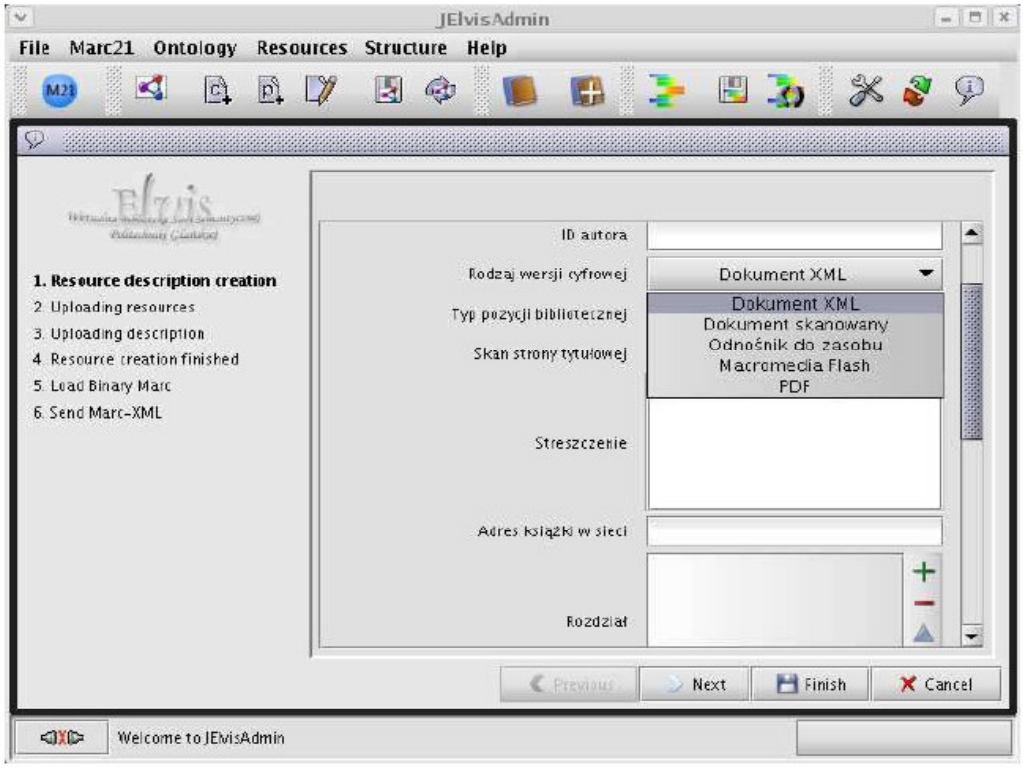


Figura 34: JElvisAdmin

La pubblicazione di un documento utilizzando il tool JElvisAdmin può essere diviso in alcune fasi:

1) Preparazione: l'utente crea un nuovo documento del tipo desiderato
2) Presentazione del contenuto: vengono create la struttura del documento (capitoli, range delle pagine, descrizioni delle parti) e le risorse appropriate, che contengono i dati
3) Invio dei file: il documento, con tutti i file e le descrizioni, viene inviato al server
4) Creazione della descrizione MARC21: la descrizione del documento nel formato MARC21 può essere aggiunto ad un documento già presente nel database.

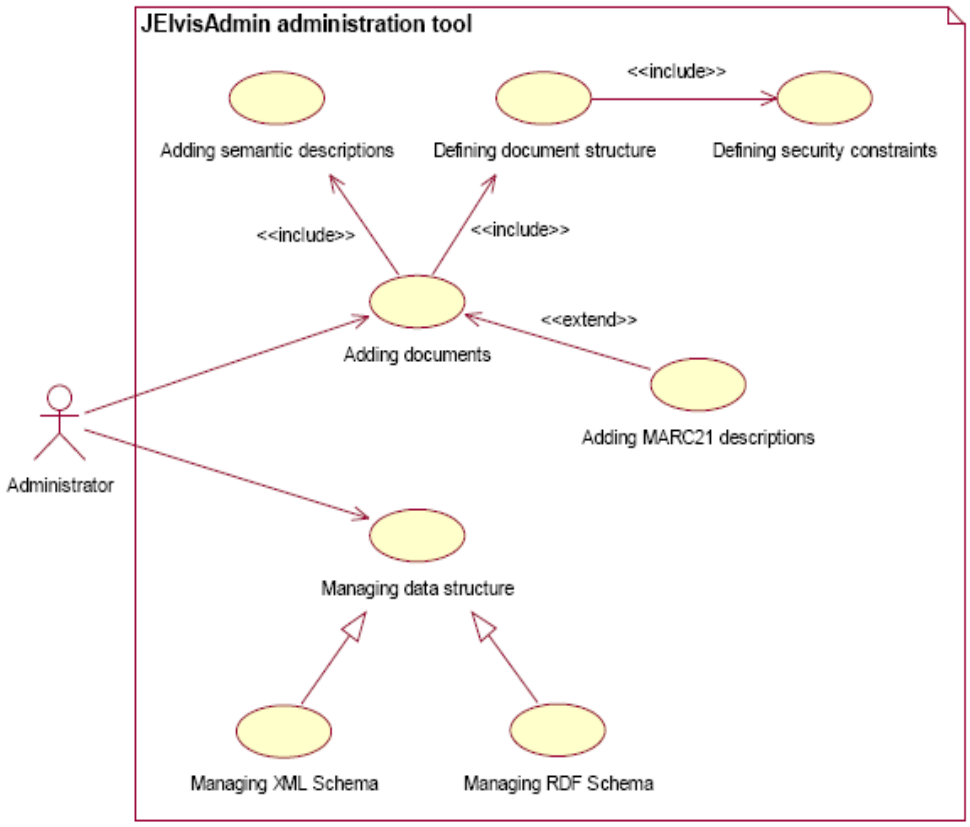


Figura 35: Use Case Diagram di JElvisAdmin

Per facilitare la pubblicazione di nuovi documenti, JElvisAdmin fornisce due ulteriori strumenti:

- **XML Schema editor**: viene utilizzato per gestire lo Schema globale XML, che descrive la struttura dei documenti nel database di Elvis. L'utente può facilmente aggiungere o rimuovere i componenti utilizzando una struttura ad albero. Ogni istanza dell'albero è descritta, tra gli altri, da un nome, una label, una descrizione e il trattamento dell'indicizzazione full-text.

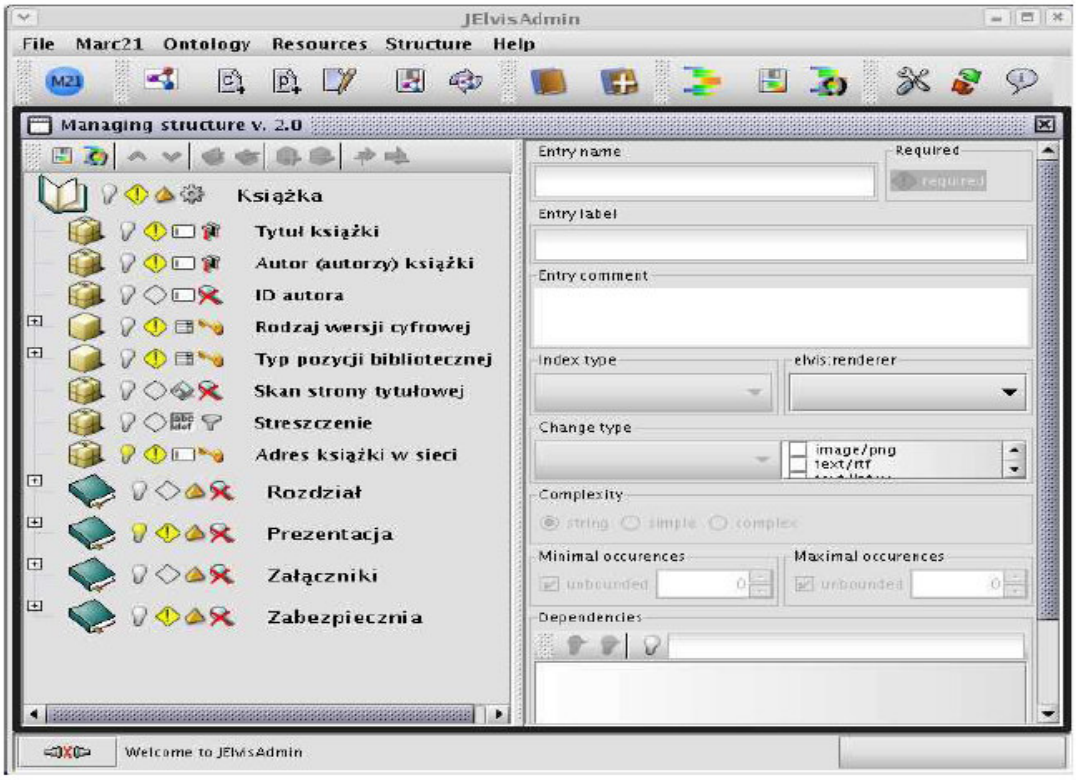


Figura 36: XML Schema editor

- **RDF Schema editor**: viene utilizzato per gestire l'ontologia ElvisOnt. L'amministratore può modificare classi esistenti, assegnare domini, modificare la classe dei dati. Anch'esso, come XML Schema editor, fa uso di una struttura ad albero.

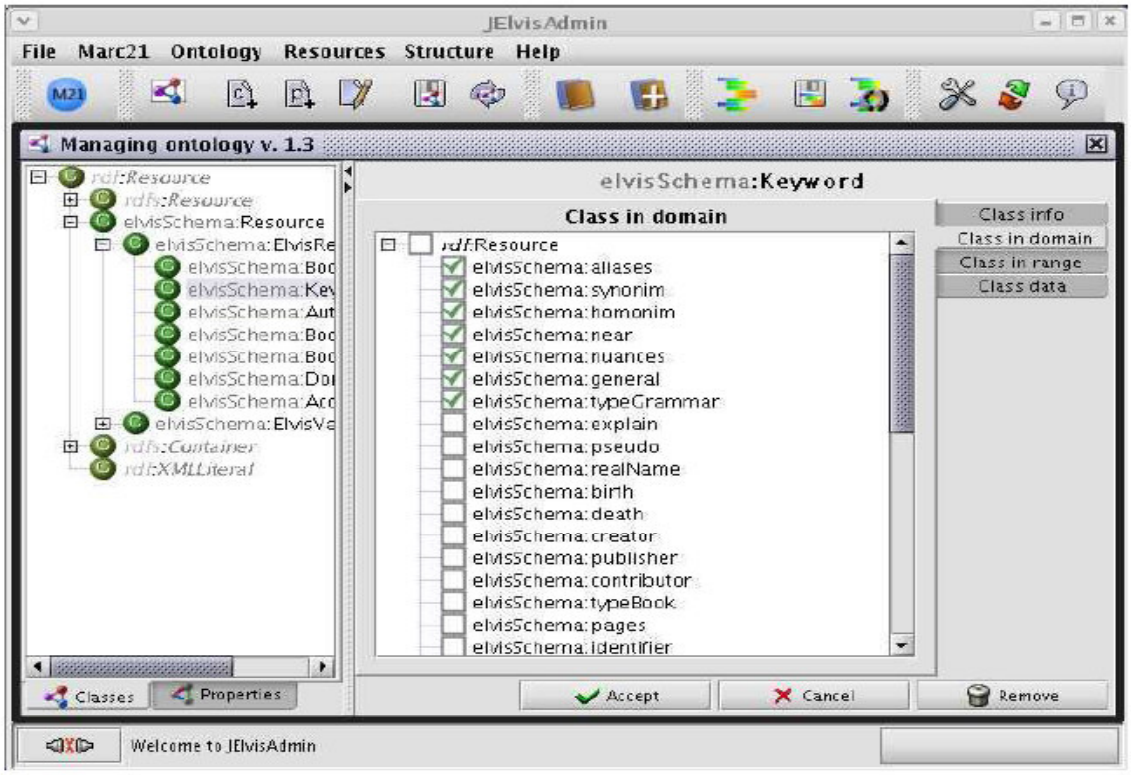


Figura 37: RDF Schema editor

La comunicazione L2L (Library-To-Library) è una nuova funzionalità delle moderne librerie virtuali. L'idea è quella di riunire un certo numero di librerie virtuali in un'unica e grande libreria attraverso protocolli di comunicazioni L2L. In questo modo le librerie condividono le loro risorse consentendo ai loro utenti di accedere ai documenti di altre librerie.

Quando un utente invia una query, ad esempio, una richiesta di ricerca, alla libreria virtuale, il sistema la elabora e allo stesso tempo la invia alle altre librerie, le quali la processano a loro volta. Infine, i risultati vengono inviati alla prima libreria, la quale li raggruppa, rimuove i duplicati e li mostra all'utente. Questa operazione è completamente trasparente all'utente.

Ovviamente, la comunicazione L2L presenta diversi problemi. Le librerie presentano diverse modalità di memorizzazione delle informazioni. Un utente, ad esempio, ricerca i testi scritti da Charles Dickens prima del 1850, ma alcune librerie potrebbero

non memorizzare l'informazione riguardo alla data di pubblicazione dei libri, di conseguenza la ricerca non avrebbe dei risultati corretti.
La libreria Elvis comunica con le altre librerie attraverso il ELP (Elvis Library Protocol). E' un nuovo protocollo ed è stato realizzato specificatamente per Elvis. Esso presenta, rispetto agli altri protocolli, alcune caratteristiche uniche, fra le quali:

- essere in grado di comunicare con successo anche con le librerie, che utilizzano differenti formati di descrizione dei documenti
- essere in grado di riconoscere i duplicati di libri da diverse librerie
- facile registrazione di nuove librerie
- architettura basata sul P2P senza colli di bottiglia
- comunicazione attraverso i messaggi SOAP

Il protocollo ELP è stato esaminato utilizzando 3 "cloni" della libreria Elvis. Ognuno di essi presentava un differente formato di descrizione. E' stato così dimostrato che il protocollo consente un efficace utilizzo delle risorse di tutte le librerie, propagando correttamente le query di ricerca e raggruppando i risultati.
La seguente figura mostra come la query dell'utente venga elaborata da un sistema L2L (frecce rosse) e come sia ottenuta e presentata la risposta (frecce viola).

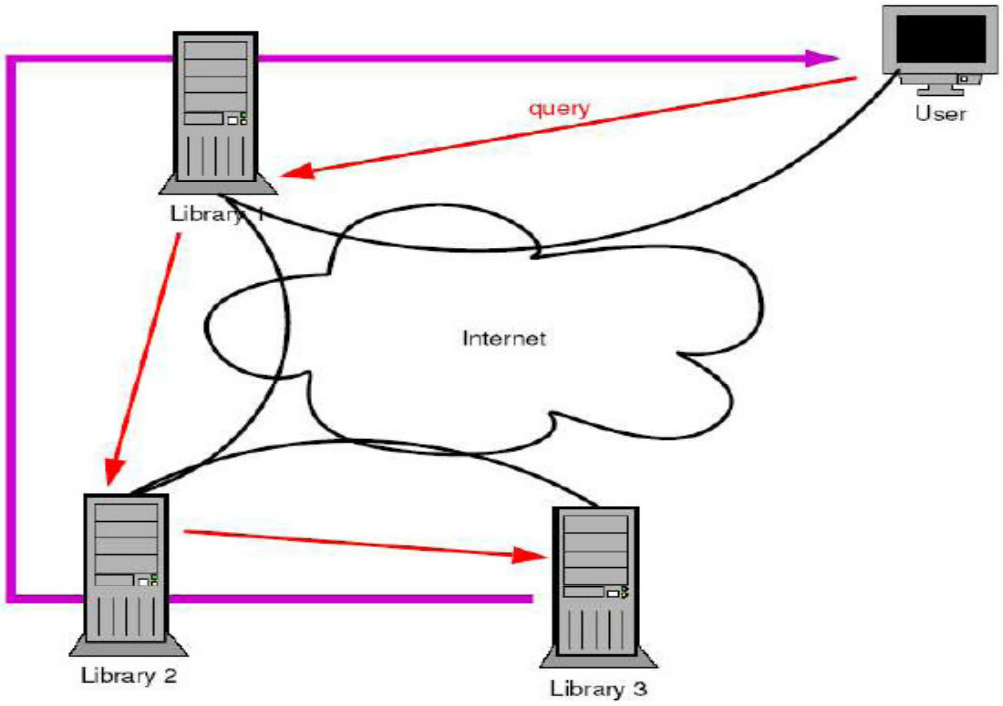


Figura 38: Comunicazione L2L

Il principale problema della comunicazione fra due applicazioni Web è il fatto che entrambe debbano utilizzare esattamente lo stesso protocollo per scambiare le informazioni. Un solo byte sbagliato comporterà la non comprensione del messaggio. SOAP (Simple Object Access Protocol) rappresenta il tentativo di creare un comune mezzo di comunicazione fra due applicazioni Web. I messaggi sono costituiti da due parti: SOAP Header e SOAP Body. La prima fornisce le informazioni riguardo i namespace, la seconda parte può contenere ogni tipo di contenuto in un formato XML. Il vantaggio di questo tipo di messaggio è che può essere ricevuto da ogni applicazione conforme al protocollo SOAP. In base alla capacità dell'applicazione, può essere interpretata e usata una parte o l'intera informazione. Elvis utilizza SOAP nel protocollo ELP, come strumento di comunicazione. Grazie a SOAP, nell'esperimento citato prima sui 3 "cloni" di Elvis, si è dimostrato che le librerie sono in grado di scambiare con successo le informazioni riguardo le loro risorse senza utilizzare un unico e preciso protocollo.

E’ stato dimostrato che alcune risorse sono più richieste di altre dagli utenti. Diventa necessario, allora, consentire un accesso più veloce a questi documenti attraverso una memoria cache. Per questo scopo, Elvis utilizza la libreria ResourcePocket Java (http://resourcepocket.sf.net), che incrementa le prestazioni e l’efficienza del sistema. Anche i risultati della trasformazioni XSLT vengono inseriti nella memoria cache per evitare che il sistema ripeta continuamente le stesse operazioni.

3.1.5 FOAF

L’obiettivo del progetto **Friend Of A Friend (FOAF)** [22] è di creare un Web costituito da home-page che descrivano le persone, dai collegamenti fra gli individui e da tutte le attività compiute dalle stesse persone. L’ontologia FOAF viene descritta attraverso la Ontology Web Language (OWL). Per entrare nel mondo FOAF, un utente deve generare un profilo, con il quale descrive se stesso rispettando alcune regole e indicazioni, e lo deve pubblicare nel Web. Il profilo deve attenersi alla ontologia, può essere generato a mano oppure, come accade molto spesso, copiando e modificando il profilo di un utente, già all’interno del mondo FOAF. Alcune parti di un profilo utente sono mostrate, come esempio, in Figura 42.

```
<foaf:Person>

 <foaf:mbox rdf:resource="mailto:ggrimnes@csd.abdn.ac.uk" />
 <foaf:name>Gunnar AAstrand Grimnes</foaf:name>
 <foaf:homepage rdf:resource="http://www.csd.abdn.ac.uk/~ggrimnes" />
 <foaf:workplaceHomepage rdf:resource="http://www.csd.abdn.ac.uk"/>
 <foaf:projectHomepage rdf:resource="http://www.csd.abdn.ac.uk/research/agentcities"/>
 <foaf:groupHomepage rdf:resource="http://www.csd.abdn.ac.uk/research/agentsgroup" />
 <foaf:phone rdf:resource="tel:+441224272835" />

 <foaf:depiction rdf:resource="http://www.csd.abdn.ac.uk/~ggrimnes/gfx/me.jpg" />

 <foaf:interest rdf:resource="http://www.w3.org/2001/sw/" />
 <foaf:interest rdf:resource="http://www.agentcities.net" />

 <foaf:made rdf:resource="http://www.csd.abdn.ac.uk/research/AgentCities/GraniteNights" />

 <contact:nearestAirport>
  <airport:Airport rdf:about='http://www.daml.org/cgi-bin/airport?ABZ' />
 </contact:nearestAirport>

 <foaf:knows><foaf:Person>
     <foaf:mbox rdf:resource="mailto:maym@foobar.lu" />
     <rdfs:seeAlso rdf:resource="http://martinmay.net/foaf.rdf"/>
  </foaf:Person></foaf:knows>
 <foaf:knows><foaf:Person>
     <foaf:mbox rdf:resource="mailto:apreece@csd.abdn.ac.uk" />
  </foaf:Person></foaf:knows>
 <foaf:knows><foaf:Person>
     <foaf:mbox rdf:resource="mailto:pedwards@csd.abdn.ac.uk" />
  </foaf:Person></foaf:knows>
 <foaf:knows>
  <foaf:Person foaf:name="Sonja A Schramm">
     <foaf:mbox_sha1sum>
      83276f91273f2900cf0b6657b3708b736276ef81
     </foaf:mbox_sha1sum></foaf:Person>
 </foaf:knows>

 <rdfs:seeAlso rdf:resource="http://www.csd.abdn.ac.uk/~ggrimnes/codepict.rdf" />
 <rdfs:seeAlso rdf:resource="http://www.csd.abdn.ac.uk/research/agentsgroup/foaf.rdf" />

</foaf:Person>

<rdf:Description rdf:about="">
 <wot:assurance rdf:resource="foaf.rdf.asc" />
</rdf:Description>
```

Figura 39: Parti di un file FOAF

Questo esempio mostra alcuni aspetti importanti di FOAF:

- La proprietà *<foaf:knows>* punta ad altre persone conosciute dall'utente stesso, creando, così, una comunità in rete
- Nel mondo FOAF, gli utenti non hanno bisogno di un URI, perché sono identificati attraverso la loro *<foaf:mbox>* (oppure *<foaf:mbox_sha1sum*), ad esempio, l'indirizzo e-mail.

- Le proprietà <*foaf:knows*> non assumono il valore dell'URI di altre persone, ma ountano ad un anonimo nodo RDF del tipo <*foaf:Person*>, che contiene la <*foaf:mbox*> della altra persona.
- <*foaf:mbox_sha1sum*> viene utilizzata per nascondere l'indirizzo e-mail per motivi di privacy.
- Altri file di FOAF sono collegati attraverso <*rdfs:seeAlso*>
- La proprietà <*wot:assurance*> punta ad una sigla di questo file, utilizzando la chiave PGP della persona e fornendo una modalità sicura per conoscere colui che ha scritto le istruzioni.

Il progetto FOAF è nato nel 1999, ma ha guadagnato un interesse significativo solo negli ultimi due anni, dovuto all'incremento degli studi riguardo il Semantic Web. Dal Settembre 2003, sono iniziati degli esperimenti su 9097 nodi, che rappresentano altrettante persone. I dati sono costituiti da 147527 triple, ci sono 1066 proprietà distinte, ma solo 116 vengono utilizzate più di 100 volte.
All'interno del grafico FOAF, ognuno può solitamente identificare gruppi di persone, che sono molto "vicine" nella vita reale. Ad esempio, gli individui che appartengono ad un singolo gruppo di ricerca. In un gruppo di questo tipo ci sono molti collegamenti <*foaf:knows*> interconnessi fra loro, e il livello di dettaglio riguardo ogni persona è simile, perché, come scritto precedentemente, i loro profili sono spesso copiati e leggermente modificati da un utente FOAF, oppure perché sono tutti generati dalla stessa persona, oppure perché provenienti dallo stesso database. Inoltre, un gruppo di solito possiede solo una connessione diretta con il resto del grafico FOAF, o, addirittura, in certi casi, non ha nessuna connessione. Nell'esempio, visualizzato in Figura 43, possiamo notare come all'interno del gruppo dell'Aberdeen Computing Science le persone si conoscano bene, ma l'unico collegamento verso la rete FOAF avviene attraverso un unico nodo-persona.

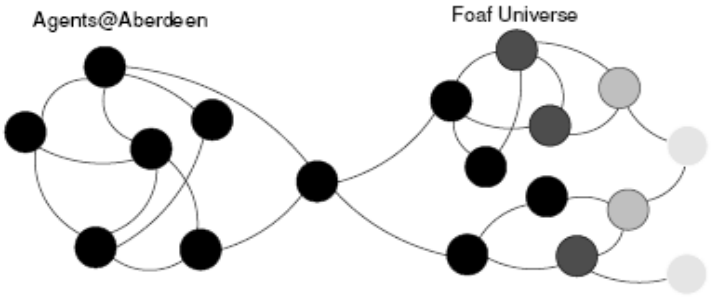


Figura 40: Un gruppo FOAF con un'unica connessione al mondo FOAF

3.1.6 FOAF-Realm

FOAF è un progetto, che fa uso dell'RDF per descrivere le relazioni di amicizia fra le persone. Si utilizzano, quindi, queste informazioni, ad esempio, per controllare l'accesso alle risorse. L'unica cosa che interessa è conoscere la distanza fra una persona e un'altra. Si può, a questo punto, descrivere come una persona può guardare, ad esempio, le nostre foto specificando la lunghezza massima del percorso, che separa me da lui o lei.

L'idea di FOAF di definire un grafico diretto e distribuito della relazioni di amicizia, dove ognuno specifica che solo il suo amico e nessun altro può modificare l'informazione, incontra qualche problema che dovrebbe essere risolto. Il primo è il problema della sicurezza, che, oltre a tutte le informazioni descritte dall'ontologia FOAF, richiede anche che sia fornito un valore SHA1 di password. Inoltre, dire che A:knows B non è abbastanza, perché nel mondo reale spesso questa relazione è espressa in maniera più precisa, come A:knows-very-good B. L'ultimo, ma non il più piccolo, è il problema della fiducia e della gestione: siccome l'informazione è distribuita, bisogna controllare che nessun estraneo, ad esempio, aggiunga nuove relazioni di amicizia e, in qualche modo, violi i vincoli di sicurezza imposti dal possessore della risorsa guadagnando l'accesso alla risorsa stessa.

E' molto comune, nel mondo reale, dire che uno dei nostri amici è più intimo degli altri. In molti casi, valutiamo le nostre amicizie in base agli eventi accaduti finora. Per questo motivo, esistono delle estensioni di FOAF, che forniscono proprietà

addizionali utili per dire, ad esempio, che qualcuno è un nostro "grande amico" oppure "non l'ho mai incontrato".

L'intuizione di **FOAF-Realm** [23] è di trattare le situazioni del mondo reale in maniera simile. Ad esempio, in alcuni casi, noi vorremmo condividere alcune risorse con amici dei nostri amici piuttosto che con i nostri stessi amici. Se abbiamo un grande amico, ciò significa che i suoi grandi amici sono conosciuti meglio da noi rispetto ad alcuni nostri amici che conosciamo appena o che non abbiamo mai incontrato. Questo è il motivo perché la valutazione delle amicizia è considerata una soluzione.

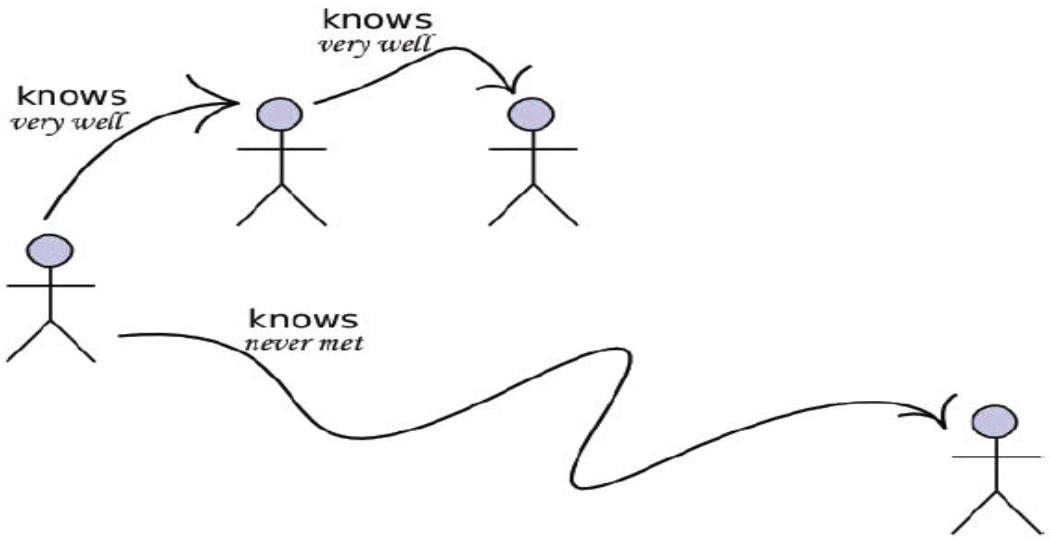


Figura 41: Chi è l'amico più intimo?

Assumendo che <foaf:knows> rappresenti l'amicizia media, c'è comunque un intero intervallo di amicizia, dal molto stretto (un grande amico) a molto distante (persona che non ho mai conosciuto). Per questo motivo, si è pensato di valutare ogni amicizia da 0% (molto distante) a 100% (molto stretto), definendo con il 50% l'amicizia media.

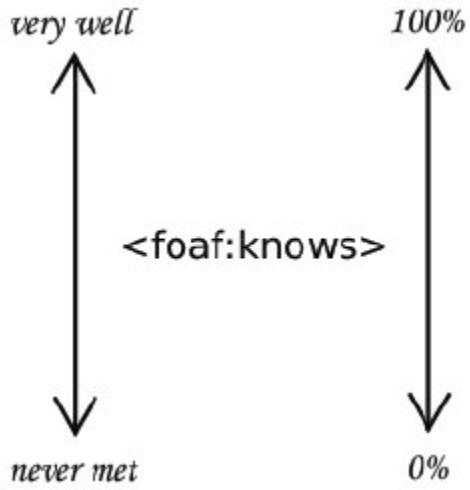


Figura 42: Valutazione su <foaf:knows>

A questo punto bisogna assegnare questa informazione alla istruzione <foaf:knows>. Siccome FOAF è un'applicazione RDF, esso presenta tutti i suoi benefici, ma anche i suoi aspetti negativi. La modalità più conveniente è di far uso delle reificazioni, quindi, inserire delle affermazioni riguardo una istruzione. La Figura 46 mostra come valutare l'amicizia fra due persone.

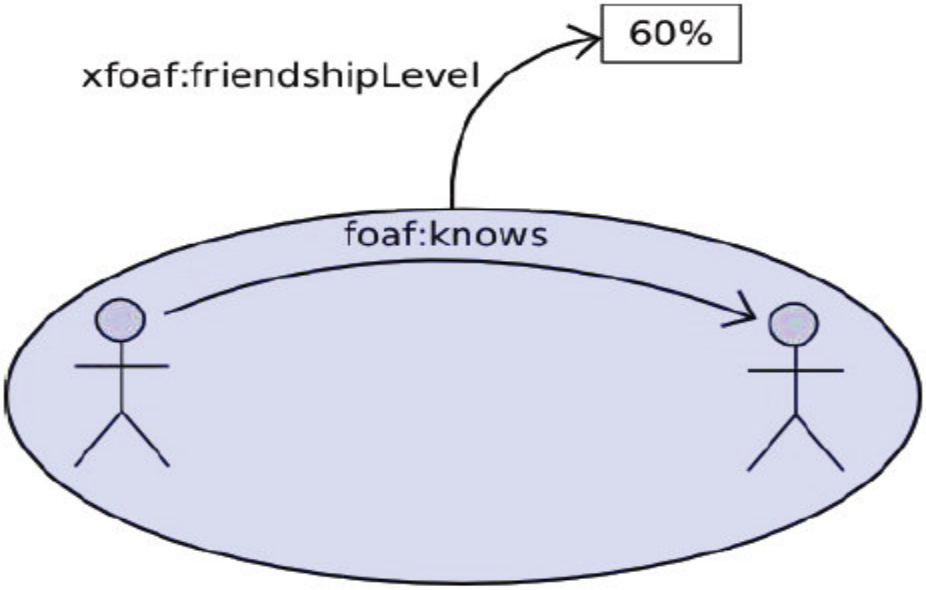


Figura 43: Istruzione <foaf:knows> reificata

Per scoprire se la persona specificata è autorizzata all'accesso della risorsa entro ruoli definiti, bisogna trovare la distanza minima fra due persone e il più alto livello di amicizia. Ci sono comunque due approcci, che dipendono dall'obiettivo da raggiungere:

- se è necessario trovare il valore esatto di distanza e il livello di amicizia fra due persone, la scelta migliore è l'algoritmo di Dijkstra
- se si conosce esattamente la massima distanza accettabile o il minimo livello accettabile di amicizia, deve essere realizzata una piccola modifica all'algoritmo, per terminare il processo di valutazione non appena è stato scelto il percorso corretto.

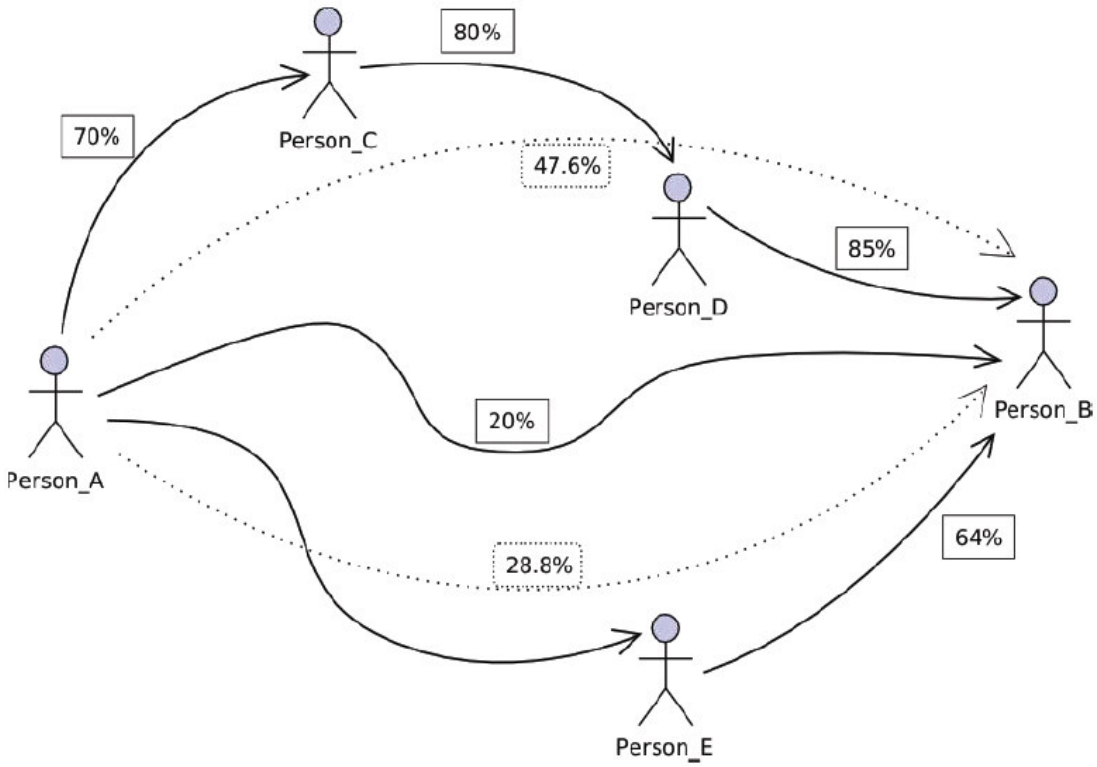


Figura 44:Valutazione dell'amicizia tra Person_A e Person_B

La Figura 47 mostra delle possibili soluzioni, dipendenti, ancora, dall'obiettivo dato. La strada più corta tra Person_A e Person_B è la connessione diretta, ma il percorso

con il più alto livello complessivo di amicizia è Person_A → Person_C → Person_D → Person_B [47,6%]. Se la richiesta prevede che vi siano al massimo due connessioni e un livello di amicizia sopra il 25%, l'algoritmo può terminare con Person_A → Person_E → Person_B [28,8%]. In molte situazioni reali questo metodo permette di risparmiare molto tempo rispondendo alla funzione isUserInRole().

Per poter essere gestita anche all'interno dell'applicazione FOAF_Realm, lo schema in linguaggio FOAF esteso, mostrato in Figura 47, necessita dell'autenticazione. La proprietà <foaf:mbox> può essere utilizzata come un login, tuttavia, in alcune situazioni, viene memorizzata solo la <foaf:mbox_sha1sum>. L'applicazione effettua, quindi, il calcolo SHA1 dal valore <foaf:mbox> e seleziona la persona con il valore risultante di <foaf:mbox_sha1sum>. La password può essere gestita allo stesso modo. La sola informazione memorizzata nel modello è il valore <xfoaf:password_sha1sum>, che è confrontato, dalla implementazione org.apache.catalina.Realm, con il calcolo SHA1 effettuato dalla password fornita dall'utente.

Il progetto FOAF-Realm è nato proprio mentre è stata ultimata l'ultima implementazione di Sevlet/JSP container-Tomcat 5.0. La versione di quest'ultimo fornisce l'interfaccia org.apache.catalina.Realm per la realizzazione dei regni. Le idee fornite dal modello FOAF possono essere utilizzate in uno di questi regni, fornendo non un modo per memorizzare l'informazione di autenticazione, ma solo un nuovo approccio per autenticare le persone e definire i loro privilegi.

FOAF-Realm è costituito da tre parti, che, insieme alla libreria Jena e al database HSQL, sono disposte sui differenti livelli dell'architettura.

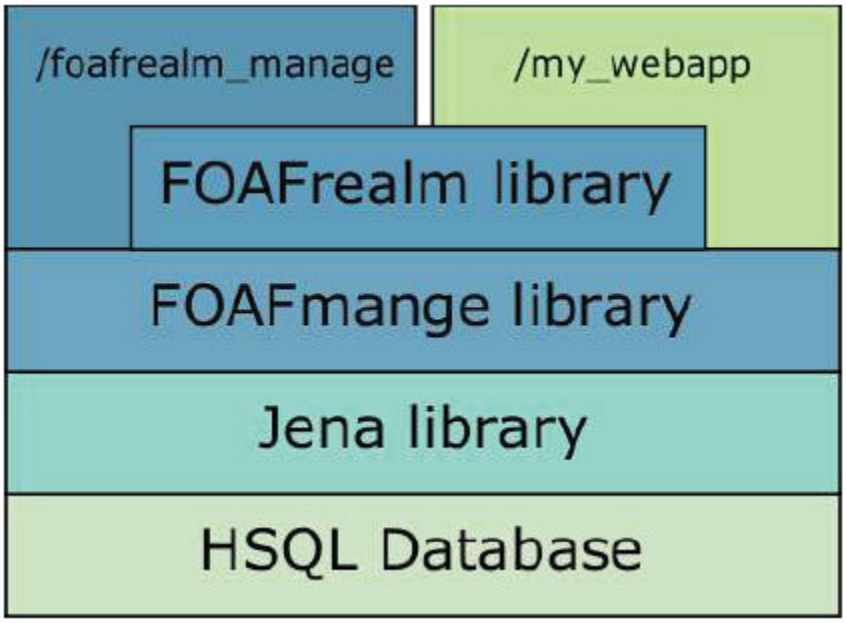


Figura 45: Architettura FOAF-Realm

La libreria **FOAF-manage** stabilisce una comunicazione con la libreria Jena per poter gestire il modello con l'informazione FOAF arricchita dalle reificazioni riguardo il livello di amicizia, definito dalle istruzioni <foaf:knows>, e dai calcoli SHA1 delle password di persone definite nel modello. La libreria è anche responsabile della valutazione dell'amicizia fra due persone, occupandosi, per quanto riguarda la scelta del percorso, sia dell'algoritmo di Dijkstra sia della piccola modifica dello stesso algoritmo, di cui abbiamo parlato in precedenza.

La libreria viene utilizzata direttamente dal **FOAFrealm_manager**, una applicazione Web, che consente di inserire o modificare le persone e le loro connessioni <foaf:knows> nel modello FOAF esteso. In aggiunta viene fornita una pagina distance.jsp per consentire il controllo delle distanze fra le persone.

La Figura 49 mostra le classi e le funzioni principali implementate nella libreria FOAFmanage.

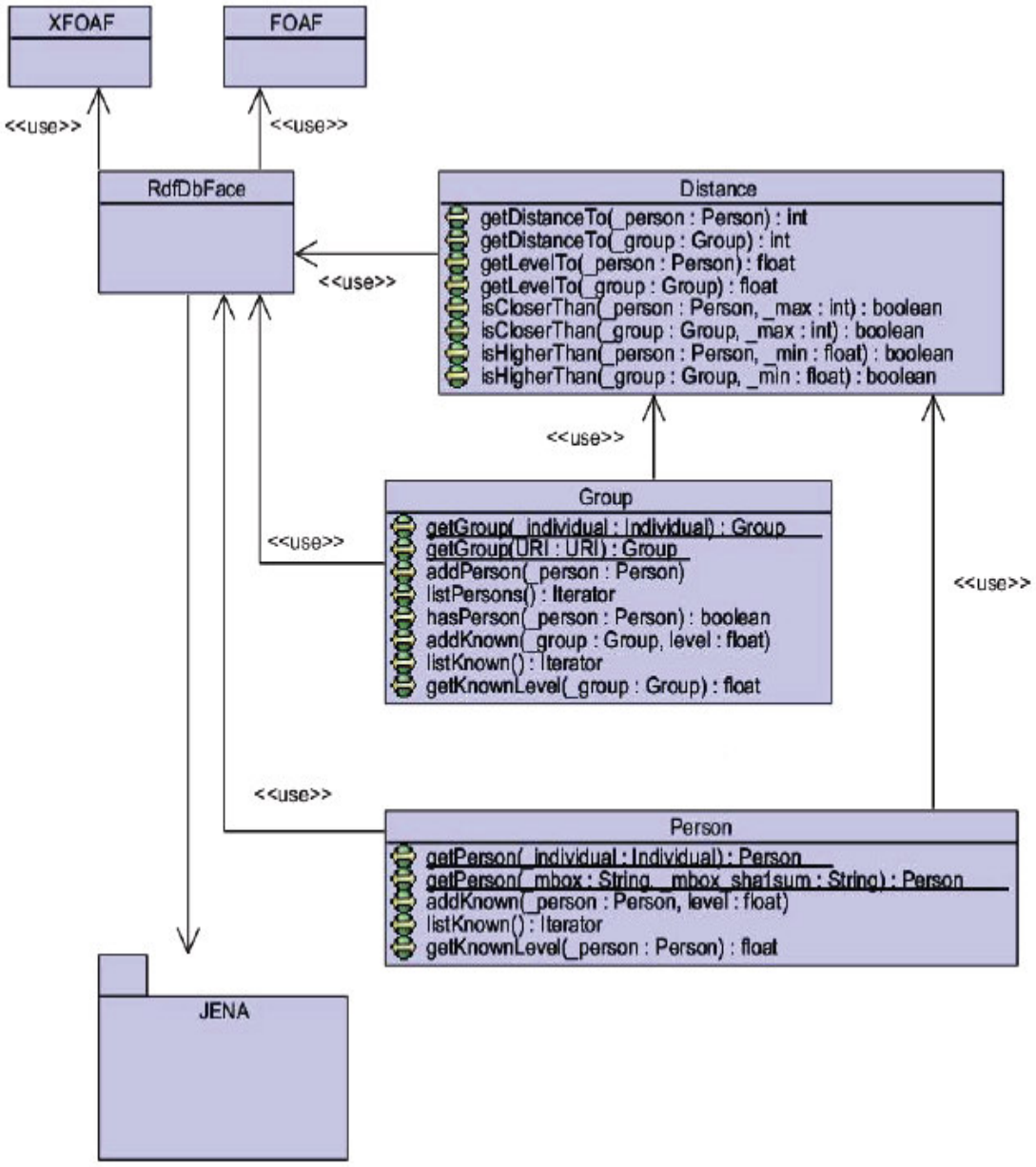


Figura 46: Class Diagram della libreria FOAFmanage

La libreria **FOAFrealm** è costituita dalle implementazioni di org.apache.catalina.Realm e di java.security.Principal insieme alle classi aggiuntive per la esecuzione dell'autenticazione e per la valutazione delle espressioni del regno. Il meccanismo di autenticazione è basato solamente sulle password SHA1, mentre le definizioni riguardanti il regno possono essere abbastanza complicate.

La Figura 50 mostra le classi e le funzioni principali implementate nella libreria FOAFrealm.

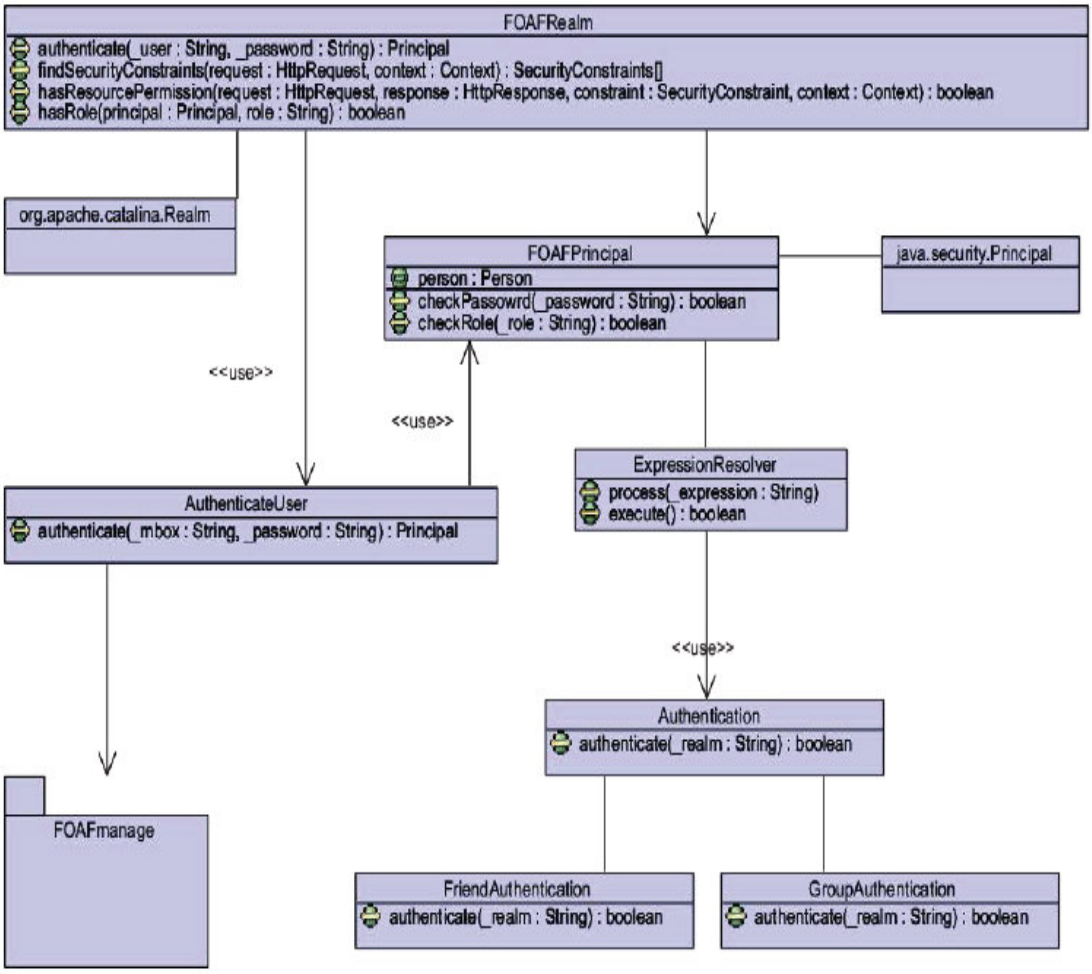


Figura 47: Class Diagram della libreria FOAFrealm

Come abbiamo scritto precedentemente, le espressioni per la definizione dei ruoli possono essere abbastanza complicate. Ognuna di esse può consistere di:

- Definizioni di distanze di amicizia e di livelli: ***F[mbox]distance{.\|,}level*** , dove *distance* è il numero minimo di connessioni tra *mbox* e la persona divenuta autorizzata, *level* è il valore frazionario di minimo livello di amicizia. Dipendendo dall'uso di .[punto] o ,[virgola] , entrambi o almeno uno dei vincoli deve essere conosciuto

- Definizioni di distanze simili di gruppo e di livelli: ***G[group_uri]distance{.\,}level*** , dove *distance* è il numero massimo di connessioni fra *group_uri* e un gruppo/comunità, di cui la persona divenuta autorizzata è membro, *level* è il valore frazionario di minimo livello di amicizia. Dipendendo dall'uso di .[punto] o ,[virgola] , entrambi o almeno uno dei vincoli deve essere conosciuto

In aggiunta, ognuna delle definizioni può essere raggruppata in vincoli, il quali potrebbero imporre che:

- ogni definizione deve essere conosciuta: **&(D1, D2, D3, …)**
- almeno una delle definizioni deve essere conosciuta: **|(D1, D2, D3, …)**
- esattamente una delle definizioni deve essere conosciuta: **^(D1, D2, D3, …)**
- nessuna delle definizioni dovrebbe essere conosciuta: **!(D1, D2, D3, …)**

Con l'uso di un tale linguaggio di espressioni, l'utente può liberamente definire ogni possibile tipo di vincolo di regno.
Per misurare la distanza fra una persona e un'altra, deve creata una istanza della classe Distance. Ogni istanza è fortemente connessa con una istanza di Persona e le successive query, riguardanti le distanze, vengono memorizzate in una memoria cache allo scopo di velocizzare il processo di misurazione. Anche se, in ogni momento, nel modello FOAF esteso viene definita o modificata una nuova amicizia, la cache viene automaticamente svuotata. La misura (distanza, livello) viene memorizzata solo quando è stato eseguito tutto l'algoritmo di Dijkstra. Tutto questo permette di evitare futuri problemi con query con un ruolo molto simile, ma con differenti misure.
FOAF-Realm può essere utilizzato in due applicazioni:

A) Condividere le proprie annotazioni in una libreria digitale

In una grande libreria digitale, uno strumento molto utile è l'abilità di selezionare i libri, o le parti di libri, più interessanti per una futura lettura. Inoltre, ad alcuni utenti

piace annotare i libri per facilitare i loro lavori o ricerche. Sarebbe, quindi, molto utile se gli utenti potessero condividere le loro selezioni e annotazioni fra i loro amici. FOAF-Realm è stato realizzato per permettere tutto questo. L'utente, infatti, può descrivere il ruolo che deve avere colui che può vedere le risorse dell'utente stesso. Attraverso le espressioni di ruolo definite in FOAF-Realm, ognuno può specificare molto precisamente chi può vedere o modificare qualche dato. La Figura 51 mostra come può la Person_A permettere alla Person_B e alla Person_C di ottenere l'accesso alle sue risorse, mantenendo i suoi stessi dati protetti dagli utenti non autorizzati.

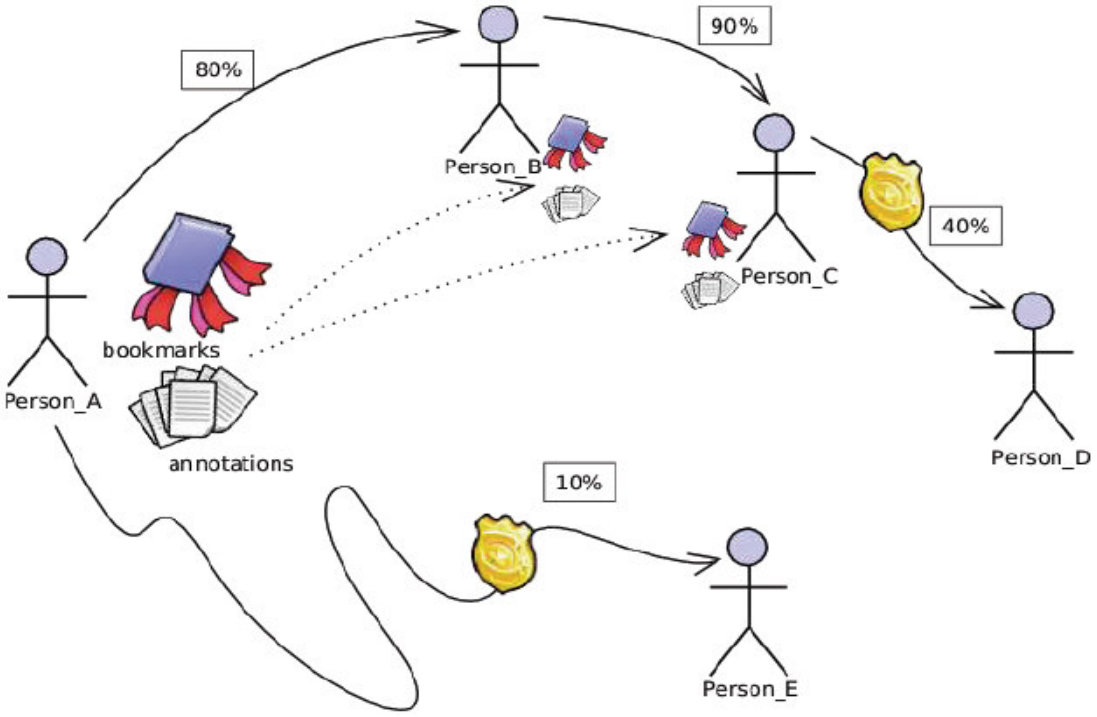


Figura 48: Condivisione delle annotazioni con gli amici più stretti

B) Conoscere gli amici per effettuare ricerche migliori

La libreria digitale Elvis realizza sofisticati algoritmi di ricerca per fornire all'utente i risultati migliori possibili riguardo le loro ricerche. La gestione della libreria è basata su:

- un indice su tutto il testo
- una descrizione semantica delle risorse
- una memorizzazione delle attività effettuate

I risultati di alcune ricerche hanno dimostrato che l'utilizzo di almeno due dei punti precedenti permette un miglioramento di circa il 40% sulla qualità del processo di ricerca.

Siccome non è facile fornire una soddisfacente descrizione semantica della risorsa, l'informazione riguardo le attività compiute finora dall'utente può essere molto importante per il processo di ricerca. Riguardo i nuovi utenti o quelli che utilizzano poco le funzionalità della libreria è difficile conoscere le loro preferenze. Si può, quindi, chiedere direttamente ai loro amici di realizzare un loro profilo utente dell'amico, fissando, comunque, dei vincoli e mantenendo una certa cautela sui dati raccolti. Ovviamente, si è assunto che la maggior parte delle persone possiede amici con interessi simili.

CONCLUSIONI E LAVORO FUTURO

In questa tesi è stata affrontata l'analisi di architetture Peer to Peer (P2P) utili per la gestione di database distribuiti.

Inizialmente, nel Capitolo 1, per introdurmi nell'argomento, sono state descritte le principali topologie di reti, di base e ibride, e le applicazioni P2P più utilizzate dagli utenti: **Gnutella1**, **Gnutella2** e **FreeNet**, esempi di reti decentralizzate; **Napster**, esempio di rete ibrida centralizzata a cluster; **KaZaA**, esempio di rete ibrida decentralizzata a cluster, riguardo la quale è stato presentato lo studio di misurazione effettuato dai ricercatori J.Liang, R.Kumar, K.W.Ross, presso il Dipartimento di Computer and Information Science Polytechnic dell'Università di Brooklyn, USA.

Nel Capitolo 2 è stato descritto come avviene la ricerca di un file all'interno di un'architettura P2P, il sistema **Chord**, in cui i nodi sono disposti ad anello. In primo luogo, sono stati definiti gli elementi principali della rete (Chiave k, Valore e il NodeID) e le 5 operazioni fondamentali, soffermandoci sulle funzioni di ricerca, *lookup(k)*, e di ingresso e uscita di un nodo n dalla rete, rispettivamente *join(n)* e *leave( )*. In particolare, riguardo queste ultime due funzioni, è stato descritto un esempio per capire in che modo la rete si adatta all'ingresso o all'uscita dei nodi da essa. Successivamente è stato presentato il progetto Chord, realizzato dal gruppo Parallel & Distributed Operative Systems (PDOS), descrivendo i passi per effettuare e completare l'installazione di una versione del sistema sul nostro personal computer.

Nel Capitolo 3 è stata studiata e analizzata una recente architettura P2P, **HyperCuP**, in cui i nodi sono disposti ad ipercubo, o hypercube. Inizialmente, sono state elencate le caratteristiche della rete, poi, attraverso un esempio, sono state descritte le fasi per la costruzione e il mantenimento delle rete. In seguito, è stato presentato una recente estensione della topologia: un routing basato sull'ontologia. Le informazioni, fornite dai peer, vengono organizzate in categorie allo scopo di esprimere concetti generali, i quali sono, a loro volta, raccolti in un'ontologia globale. Per questo motivo, non si utilizza un unico hypercube per l'intera rete, ma un hypercube per ogni concetto, consentendo, in questo modo, alla rete di supportare combinazioni logiche dei

concetti dell'ontologia durante una query. Questo è indubbiamente un argomento molto interessante nell'ambito dei Peer Data Management System (PDMS). Per comprendere meglio il processo di routing, sono stati presentati due esempi, mostrando come i peer, con interessi e servizi simili, vengano raggruppati in cluster, i quali sono assegnati ad una specifica logica combinazione di concetti ontologici. Successivamente, è stata descritta una implementazione di HyperCuP, realizzata dai ricercatori P. Bugalski, S. Grzonkowski, presso il Digital Enterprise Research Institute (DERI), presentando la sua architettura a due livelli, le funzioni implementate e quelle ancora da realizzare. La Elvis Digital Library è un'applicazione che utilizza HyperCuP nella comunicazione fra le librerie, Library to Library (L2L), presenti nel Web: per effettuare, ad esempio, l'ingresso di una nuova libreria in una rete esistente, per gestire l'invio di una query nella rete L2L, per controllare l'autenticazione e l'autorizzazione degli utenti che vogliono ottenere risorse da una specifica libreria. Inoltre la libreria Elvis risulta essere interessante, perché, oltre al meccanismo di ricerca semplice, consente anche una ricerca semantica, durante la quale possono essere utilizzate, automaticamente, le preferenze dell'utente. In seguito è stato presentato il progetto FOAF-Realm, in cui l'accesso alle risorse di un nodo è consentito solo agli amici più stretti.

In conclusione, HyperCuP è un progetto nato nel 2004 e sembra essere molto promettente. Alcune tematiche, quindi, sulle quali, in futuro, dovrà focalizzarsi la ricerca sono: l'approfondimento sullo sviluppo e sul funzionamento di tale implementazione, installando e utilizzando una versione del programma; l'analisi delle comunicazioni e degli scambi dei dati fra i peer dell'ipercubo, verificando che, l'utilizzo del routing basato sull'ontologia, comporti, come esito di una query, risultati efficienti; lo studio riguardo la sicurezza dei dati all'interno di HyperCuP, in particolare i problemi di autenticità e di controllo di accesso ai dati.

BIBLIOGRAFIA

[1] FastTrack Homepage, http://www.fasttrack.nu, 2005

[2] FreeNet Homepage, http://freenet.souceforge.net, 2005

[3] Gnutella servers and host cache, http://www.gnutella.co.uk/servers, 2005

[4] KaZaA Homepage: http://www.kazaa.com, 2005

[5] Limewire Homepage: http://www.limewire.com, 2005

[6] Napster Homepage, http://www.napster.com, 2005

[7] OpenNapster Homepage: http://www.opennapster.org, 2005

[8] OpenP2P, http://openp2p.com, 2005

[9] P2P Sicuro, http://www.p2psicuro.it, 2005

[10] I. Clarke, O. Sandberg, B. Wiley, T.W. Hong: "FreeNet: A Distributed Anonymous Information Storage and Retrieval System", Division of Informatics, University of Edinburgh, 1999

[11] J. Liang, R. Kumar, K.W. Ross: "The KaZaA Overlay: A Measurement Study", Brooklyn, NY, USA, 2004

[12] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan: "Chord: A scalable peer-to-peer lookup service for Internet applications", Massachusetts Institute of Technology (MIT), Laboratory for Computer Science, USA, 2001

[13] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan: "Chord HOWTO", 2001

[14] M. Schlosser, M. Sintek, S. Decker, W. Nejdl: "HyperCuP – Hypercubes, Ontologies and Efficient Search on P2P Networks", Computer Science Department, Stanford University, 2004

[15] M. Schlosser, M. Sintek, S. Decker, W. Nejdl: "HyperCuP – Shaping Up Peer-to-Peer Networks", Computer Science Department, Stanford University, 2004

[16] M. Schlosser, M. Sintek, S. Decker, W. Nejdl: "A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services", Computer Science Department, Stanford University, 2004

[17] P. Bugalski, S. Grzonkowski: “HyperCuP State of Art, implementation analysys”, Digital Enterprise Research Institute (DERI), National University of Ireland, Galway, Ireland, 2005

[18] P. Bugalski, S. Grzonkowski: “HyperCuP – Early alpha version”, Digital Enterprise Research Institute (DERI), National University of Ireland, Galway, Ireland, 2005

[19] M. Synak, S.R. Kruk: “Elvis Digital Library – e-Library with Semantics”, Digital Enterprise Research Institute (DERI), National University of Ireland, Galway, Ireland, 2004

[20] S.R. Kruk, S. Decker, L. Zieborak: “JeromeDL – Adding Semantic Web Technologies to Digital Libraries”, Digital Enterprise Research Institute (DERI), National University of Ireland, Galway, Ireland, 2004

[21] S.R. Kruk: “Advanced search and browsing in digital libraries”, Digital Enterprise Research Institute (DERI), National University of Ireland, Galway, Ireland, 2004

[22] G. Grimnes, P. Edwards, A. Preece: “Learning Meta-Descriptions of the FOAF Network”, Computing Science Dept. King’s College, University of Aberdeen, Scotland, 2004

[23] S.R. Kruk: “FOAF-Realm – control your friends’ access to resources”, Digital Enterprise Research Institute (DERI), National University of Ireland, Galway, Ireland, 2004