Università degli Studi di Parma

Dipartimento di Informatica

Reti di Calcolatori

Peer to Peer Routing Protocol: un protocollo di routing per reti P2P

Autori:
Alessio Bortolotti
Jacopo Freddi

Professore:
Roberto Alfieri

21 luglio 2014

Indice

Sommario

Scopo del progetto è l'elaborazione di un algoritmo che permetta ad un insieme di nodi di connettersi via `TCP` e gestire la rete senza l'ausilio di un server. Date le grandi affinità con la struttura dei router, il protocollo elaborato è stato chiamato `PRP`, ovvero Peer-to-Peer Routing Protocol.
Date le complessità di implementazione, la soluzione proposta nel codice è necessariamente incompleta. Gli scopi di questa relazione sono:

1. mostrare l'architettura di rete risultante dall'utilizzo dell'algoritmo e metterne in evidenza alcune proprietà;
2. illustrare il funzionamento dell'algoritmo;
3. discutere dei miglioramenti che possono essere apportati all'algoritmo.

Notare che, anche se l'applicativo di esempio sopra modellato è una semplice chat, il protocollo può essere utilizzato anche per altri scopi, quali ad esempio:

1. trasferimento di file;
2. mail service;
3. `VoIP`.

1 Requisiti

È dato un insieme di nodi, collegati tra loro tramite il protocollo `TCP`. Ogni nodo simula un router: tiene aperte le connessioni verso gli altri nodi ed inoltra i pacchetti in arrivo verso il router interessato. È richiesto un protocollo che mantenga allineate le tabelle di inoltro dei nodi e risponda automaticamente agli errori ed ai cambiamenti di topologia della rete.

2 L'algoritmo

L'algoritmo progettato è semplice ma robusto. Per la procedura di prima connessione è richiesto specificare manualmente uno o più nodi a cui connettersi, per le connessioni successive può essere utile salvarsi in un file (o in un database) una lista contenente indirizzi e porte di nodi noti, in modo da rendere più efficace la procedura di connessione. Nel caso i nodi debbano simulare una rete di router sempre attivi, comunque, l'eventualità di doversi connettere nuovamente è molto remota e le problematiche relative ad eventuali cambiamenti di indirizzo/porta dei nodi e dell'identità di questi sono molto semplificate dato che reti del genere sono generalmente progettate per essere robuste, stabili e sempre attive.

2.1 Pseudocodice dell'algoritmo

```
Seleziona un nickname ed una porta di ascolto disponibile;
Seleziona un nodo all'interno della rete;
Connettiti a quel nodo;
```

```
Comunica a quel nodo le tue informazioni di ascolto ed il tuo nickname;
Ricevi da quel nodo le informazioni relative ai nickname che possono essere
raggiunti attraverso di lui;
Ricevi da quel nodo il nodo di backup;
Inizia ad accettare connessioni dalla porta di ascolto;
In parallelo:

  1. gestisci le connessioni in entrata;

  2. gestisci i messaggi in entrata;

  3. gestisci i cambiamenti di topologia;
```

Come è possibile notare, la struttura non è poi molto diversa da una normale chat. Le gestioni in parallelo sono descritte poco più avanti, ma per ora è necessario soffermarsi su un punto critico del protocollo: il nodo di backup.

2.2 L'importanza del nodo di backup

Perché la topologia di rete resista ad errori fatali (disconnessioni da internet, cali di corrente, ...) è necessario che i nodi adiacenti a quello danneggiato abbiano la possibilità di tornare a connettersi alla rete. Per qualsiasi nodo, il nodo di backup viene fornito dal padre ed indica un nodo a cui è possibile connettersi in caso il padre si disconnetta. Il padre può adottare qualsiasi metodo per distribuire i nodi di backup, ma è necessario che la distribuzione non abbia cicli. Il metodo utilizzato nel codice presentato è semplice: ci si limita ad indicare come nodo di backup il proprio padre. Una soluzione alternativa consiste nell'indicare al proprio primo figlio il proprio padre, al secondo figlio il primo figlio, e così via. Nessuna di queste soluzioni genera cicli nella topologia di rete.
È possibile lavorare più finemente sotto quest'aspetto, dato che da esso dipendono molti parametri che determinano la capacità della rete di essere veloce ed adattativa. Una modifica, suggerita nelle possibili espansioni, prevede di bilanciare l'albero: una scelta che semplifica le operazioni di connessione dal punto di vista di chi si connette, ma le complica dal punto di vista di chi accetta le connessioni.

2.2.1 Bilanciare l'albero: il problema

Nel protocollo finora presentato, quando un nodo si connette deve decidere a quale nodo connettersi. Se il nodo scelto non è attivo, si può continuare a provare a connettersi ad altri nodi conosciuti (per entrare in una rete già costituita) o decidere di non connettersi a nessuno e dare la propria disponibilità a fungere da radice dell'albero.
La scelta del nodo a cui connettersi rende impossibile ottenere alberi bilanciati. Questo degrada le prestazioni: un albero con troppe diramazioni e bassa profondità causa un sovraccarico di tempo di elaborazione e memoria (necessari per memorizzare ed utilizzare le tabelle di inoltro, più lunghe), mentre un albero con poche diramazioni ma molto profondo aumenta notevolmente

il ritardo tra l'invio di un pacchetto da una foglia e l'arrivo di questo ad una foglia diversa. Occorre dunque un protocollo a parte per realizzare la gestione bilanciata dell'albero.

2.2.2 Bilanciare l'albero: la soluzione

La prima cosa da fare, se si desidera avere un albero bilanciato, è definire il numero di connessioni massimo accettato dal nodo. Questa può essere una scelta statica e comune per tutti i nodi o variabile e definita al momento di lanciare il client. Occorre inoltre modificare la procedura di accettazione delle connessioni: una volta raggiunto il numero massimo di nodi connessi, le successive richieste di connessione vengono rifiutate e reindirizzate verso la radice dell'albero. Il primo nodo, che deve conoscere la profondità di ciascuno dei sottoalberi, reindirizza quindi la richiesta sul sottoalbero meno profondo. Ogni nodo che riceve la comunicazione di richiesta di connessione dal proprio padre la accetta, se ha posto, oppure la indirizza verso il sottoalbero meno profondo. Questo protocollo consente di mantenere bilanciata la rete: se ogni nodo ha m connessioni massime, in una rete di n nodi il numero massimo di passaggi per passare un messaggio da una parte all'altra della rete è $2log_m(n)$, il che è un costo più che accettabile almeno per piccole o medie reti.

2.2.3 Bilanciare l'albero: il nodo di backup

Dove entra il nodo di backup in tutto questo? Come sempre, quando un nodo scompare. Alla disconnessione di un nodo, il suo primo figlio ne prende il posto come figlio del suo padre mentre gli altri nodi figli vengono 'adottati' da quest'ultimo. Questa operazione è abbastanza poco costosa e permette di gestire i cambi di topologia mantenendo una struttura economica dal punto di vista dello scambio dei messaggi.

2.3 Gestione delle connessioni in ingresso

Di default le connessioni in arrivo vengono automaticamente accettate, ma l'algoritmo prevede di rifiutarle dopo aver ricevuto una sequenza di messaggi non validi. Come in ogni struttura di questo tipo, alla porta di ascolto comunicata alla rete è associata una socket in attesa di richieste di connessioni. Ad ogni connessione accettata vengono generati due nuovi thread, per questo è consigliabile valutare, in un'applicazione pratica, l'applicazione del protocollo di albero bilanciato sopra descritto.
All'instaurarsi della connessione il nuovo nodo deve inviare al suo nodo padre le informazioni relative alla sua porta di ascolto, al proprio indirizzo IP ed al proprio nickname;
Dopo aver accettato la connessione e verificato che il nodo connesso stia utilizzando il protocollo giusto, il nodo padre deve inviare al nuovo nodo due messaggi:

1. Viene sempre inviato, anche se la rete è vuota, un messaggio contenente la tabella del nodo padre. Questa tabella serve come tabella iniziale per il nuovo nodo, ma sarà poi modificata in seguito agli eventuali cambiamenti di topologia;

2. Viene inviato un messaggio che specifica il nodo di backup per il nuovo nodo. Se il nuovo nodo è il primo nodo ad essere connesso al nodo padre ed il nodo padre è il primo nodo della rete, il nodo di backup coincide con il nuovo nodo.

Contemporaneamente, il padre manda a tutte le altre interfacce della sua rete un messaggio informativo, contenente le informazioni sul nuovo nodo. Come nelle reti di router, ogni nodo che riceve questo messaggio salva nella sua tabella di inoltro che il nuovo nodo è raggiungibile tramite il nodo da cui il messaggio è arrivato.

2.4 Gestione dei messaggi ricevuti

La gestione dei messaggi ricevuti è un'operazione fondamentale: ogni nodo è responsabile dei sottoalberi che lega a suo padre sia per quanto riguarda il mantenimento delle connessione che per la propagazione dei pacchetti di alto livello.
Quella che segue è una rassegna dei tipi di messaggio, dichiarati nel modulo `Constants`:

- `MessageHelloCode`: contiene le informazioni per contattare il nodo mittente. Viene sempre trattenuto ed è trasparente all'utente, in quanto serve unicamente al mantenimento della rete. Può però avere conseguenze nell'interfaccia grafica (ad esempio rendendo noto all'utente che il nodo è ora connesso).

- `MessagePointToPointCode`: contiene un messaggio con un singolo destinatario. Se il destinatario coincide con il client corrente il messaggio viene trattenuto, altrimenti viene inviato nuovamente verso il nodo che permette di raggiungere tale destinatario, se presente.

- `MessageBroadcastCode`: contiene un messaggio per tutta la rete. Sfruttando la peculiare disposizione della rete, è possibile gestire questo messaggio semplicemente redirigendolo verso tutti i nodi tranne quello da cui il messaggio proveniva originariamente.

- `MessageBackupNickCode`: contiene le informazioni sul nodo di backup per il mittente. Viene sempre trattenuto ed è trasparente all'utente, in quanto serve unicamente al mantenimento della rete.

- `MessageReachableCode`: segnala che un certo nodo è ora raggiungibile. Questo messaggio ha conseguenze prevalentemente logistiche e legate al mantenimento della rete e della tabella di inoltro, ma può essere utile reagire ad esso per aggiungere informazioni all'interfaccia grafica.

- `MessageNotReachableCode`: segnala che il nodo specificato non è più raggiungibile. Questo messaggio ha conseguenze prevalentemente logistiche e legate al mantenimento della rete e della tabella di inoltro, ma può essere utile reagire ad esso per aggiungere informazioni all'interfaccia grafica.

- `MessageTableCode`: segnala che il messaggio contiene la tabella del mittente, opportunamente codificata. Questo messaggio viene sempre trattenuto ed è trasparente all'utente.

2.5 Gestione dei cambiamenti di topologia

Quest'attività è fondamentale nel mantenimento della connettività intranet. I messaggi rilevanti sono pochi, ma significativi:

2.5.1 `MessageBackupNickCode`

A seconda dell'architettura adottata, questo messaggio può essere scartato. Nell'architettura in esempio, in cui il nodo di backup è sempre il proprio padre, questo messaggio viene accettato solo se è il proprio padre ad inviarlo. Quando un client accetta questo messaggio, si salva le informazioni sul nodo di backup indicato. Queste informazioni sono relative unicamente al nodo dal quale questo messaggio proviene.

2.5.2 `MessageReachableCode`

A questo messaggio è quasi sempre associata una nuova connessione. Questo messaggio viene propagato in broadcast in modo che l'informazione diventi accessibile a tutti i nodi della rete. Il messaggio prevede di inviare solo le informazioni relative al nodo, in quanto il modo in cui raggiungerlo (l'interfaccia di uscita) è implicito nel mittente. In altre parole, se `A` è connesso a `B` e `C` si connette a `B`, `C` invierà a `B` un messaggio del tipo `Hello C`, e `B` da questo messaggio capirà che `C` è raggiungibile. Quando `B` invia ad `A` il messaggio `Reachable C`, `A` scopre che `C` è raggiungibile tramite `B`. Inoltre, quando `A` propagherà l'informazione agli altri nodi, questi sapranno che `C` è raggiungibile tramite `A`. Se qualche nodo decidesse di mandare un messaggio a `C` lo invierebbe sull'interfaccia `A` specificando `C` come destinatario. `A` lo inoltrerebbe a `B` e `B` lo inoltrerebbe a `C`, in modo semplice ed immediato.
Molto spesso a questo tipo di messaggio sono associati scambi di informazioni sui nodi di backup.

2.5.3 `MessageNotReachableCode`

Il cambiamento di topologia più complesso da affrontare è accompagnato da questo messaggio. `MessageNotReachableCode` viene inviato da tutti i nodi adiacenti ad un nodo quando questo si disconnette. Essendo i nodi connessi tramite `TCP` non è necessario utilizzare messaggi di controllo (`HELLO`) per rilevare le interruzioni di una connessione: la socket rileva automaticamente che l'altro capo del canale di comunicazione è irraggiungibile ed agisce di conseguenza.
I primi vicini del nodo disconnesso, dopo aver propagato ad ogni vicino rimanente il messaggio di disconnessione, si connettono automaticamente al nodo di backup che il nodo ha fornito loro, se presente (eventualmente riutilizzando la socket appena disconnessa). Dopo questa operazione le tabelle vengono aggiornate vicendevolmente. Ogni ex primo vicino del nodo disconnesso comunica la propria tabella al nuovo nodo di backup e viceversa. Fortunatamente non è necessario stabilire un ordine di invio / ricezione delle tabelle. Ad ogni invio di tabella corrisponde infatti una propagazione della tabella aggiornata in broadcast: questa procedura, seppure poco efficiente dal punto di vista di banda utilizzata, permette molta flessibilità nell'aggiornamento delle tabelle.

Ricevuto (o generato) il messaggio di codice `MessageNotReachableCode` relativo ad un certo nodo, tutti i nodi raggiungibili tramite quello vengono eliminati dalla tabella e vengono generati altrettanti messaggi `MessageNotReachableCode` relativi ai nodi eliminati. Questi nodi per la rete non esisteranno più, finché un messaggio `MessageReachableCode` o `MessageTableCode` ne segnaleranno la disponibilità ed il nuovo percorso.
`TCP` garantisce che qualsiasi `MessageReachableCode` o `MessageTableCode` inviato dopo un `MessageNotReachableCode` arrivi nell'ordine in cui è stato inviato, vale a dire dopo il `MessageNotReachableCode`, evitando situazioni anomale in cui `MessageReachableCode` arriva prima di `MessageNotReachableCode` facendo erroneamente intendere che il nodo specificato non sia più raggiungibile.

3 Architettura di Rete

L'architettura di rete risultante dall'algoritmo è molto regolare e l'assenza di cicli è dimostrabile per induzione.
Osserveremo innanzitutto la configurazione statica della rete all'inizio del suo ciclo di vita ed in un istante generico; passeremo poi ad analizzare tutti i cambi di topologia per mostrare come nessuno di essi, partendo da una situazione senza cicli, genera cicli.

3.1 Analisi statica della rete

In generale, la topologia di rete è ad albero non bilanciato: il primo nodo ad essere online fa da padre a tutti i nodi da cui accetterà la connessione e questi ricorsivamente saranno padri dei nodi che si connetteranno a loro. Ogni nodo, al suo ingresso nella rete, si connette ad uno ed un solo nodo già appartenente alla rete: questa è la caratteristica che più di ogni altra assicura l'assenza di cicli.

3.1.1 La rete con un solo nodo

La rete iniziale ha un solo nodo attivo, in attesa di connessioni. Questa configurazione banalmente non presenta cicli.

3.1.2 La rete a regime

La rete a regime è visualizzabile come un grafo connesso aciclico o come un albero, le cui proprietà variano a seconda dell'algoritmo scelto per le procedure di connessione e riconnessione. Nel caso si voglia rappresentarla come albero, per motivi di chiarezza è opportuno adottare come radice il primo nodo della rete e come figli di un nodo tutti i nodi le cui richieste di connessione sono state accettate da questo. Questa è la configurazione che un algoritmo che implementa `PRP` dovrebbe offrire alla fine di ogni cambio di topologia.

3.2 Analisi dinamica della rete

In questa sezione vengono analizzati tutti i possibili cambiamenti di topologia e viene mostrato come (nel codice di esempio) nessuno di questi generi cicli.

3.2.1 Aggiunta di un nuovo nodo

Il nuovo nodo può richiedere la connessione ad un nodo che ospita altri figli oppure ad una foglia dell'albero. In entrambi i casi, il nuovo nodo ha solo una connessione con il nodo padre, ed ogni altro nodo della rete può raggiungere il nuovo nodo solo attraverso il padre. L'aggiunta di un nuovo nodo pertanto non introduce cicli nella struttura della rete.

3.2.2 Aggiunta di un sottoalbero

Durante le riconnessioni, è possibile che la richiesta di connessione arrivi da un nodo che fa da padre ad altri nodi. Per ipotesi induttiva il nodo ed i suoi figli sono organizzati in un grafo connesso aciclico (o albero). Il nuovo sottoalbero può richiedere la connessione ad un nodo che ospita altri figli oppure ad una foglia dell'albero. In entrambi i casi, ogni nodo del sottoalbero ha un percorso privo di cicli per arrivare alla radice del sottoalbero, e la radice del sottoalbero ha una sola connessione con il nodo padre. Dall'esterno, ogni nodo della rete può raggiungere la radice del nuovo sottoalbero solo attraverso il padre della radice (attraverso un percorso privo di cicli), e può raggiungere ogni nodo del sottoalbero tramite la radice. L'aggiunta di un sottoalbero pertanto non introduce cicli nella struttura della rete.

3.2.3 Disconnessione di un nodo

Naturalmente la disconnessione di un nodo non porta alcun ciclo nella rete. La riconnessione di un nodo alla rete cade in uno dei due casi precedenti, pertanto non introduce cicli nella struttura della rete.

4 Gli strumenti utilizzati

Il codice di esempio è stato realizzato con pochi strumenti, consigliati per lo sviluppo delle estensioni allo stesso.

4.1 Eclipse IDE for Java

Per lo sviluppo del codice è stato usato l'IDE Eclipse, gratuito e reperibile all'indirizzo www.eclipse.org. Eclipse è ottimizzato per la programmazione Java e presenta molti strumenti pensati per l'ingegnerizzazione del software, quali:

1. individuazione in tempo reale degli errori;
2. risoluzione automatizzata degli errori;

3. cambiamento automatico e project-wide di identificatori, nomi di classi e di metodi;

4. gestione semplificata di classpath, inclusioni e moduli.

Eclipse presenta, inoltre, una piattaforma di sviluppo di plugin (basata su Java) che la rende aperta all'integrazione con altri sistemi.

4.2 Git - GitHub

Lo strumento di versionamento ed il servizio di hosting sono stati entrambi affidati a GitHub (https://github.com/), una soluzione popolare per lo sviluppo collaborativo di software anche a livello Open-Source.

4.2.1 Git

Git è un sistema software di controllo di versione distribuito, creato da Linus Torvalds nel 2005. Sviluppato principalmente per monitorare ed organizzare lo sviluppo del kernel di Linux, è stato gradualmente adottato da molti come il principale strumento di versionamento grazie alla sua efficienza ed al modello centralizzato di accettazione delle modifiche, che prevede la presenza di un supervisore che decida quali accettare e quali rifiutare. Eclipse include nativamente un plugin che permette di collegarsi ad un repository Git ed effettuare le operazioni di pull, push, merge.

4.2.2 GitHub

GitHub è un servizio web di hosting per lo sviluppo di progetti software (e non solo) che usa il sistema di controllo di versione Git. Spesso considerato un rivale di SourceForge.net per l'hosting di progetti Open-Source, offre vari strumenti di gestione del repository e personalizzazione e prevede alcuni piani di sviluppo enterprise a pagamento.
Il repository del progetto è mantenuto all'indirizzo https://github.com/fagiodarkie/PRP.

4.3 Java Development Kit

Il codice utilizza la libreria Java nella versione 1.7.0-51: sebbene sia possibile che qualche classe funzioni in modo differente utilizzando altre versioni della libreria, non sono state usate feature particolari (ad esempio database). Pertanto il codice dovrebbe virtualmente funzionare con qualsiasi libreria Java che fornisca le classi necessarie (`InetAddress, Connection, Thread,` etc). Ad ogni modo, per evitare di utilizzare versioni troppo distanti tra loro si consiglia di utilizzare almeno jdk1.6.

4.4 Collaborare al progetto

Per collaborare al progetto è sufficiente installare Eclipse ed il plugin EGit, disponibile al sito www.eclipse.org/egit/. Occorre clonare il progetto dal repository specificato sopra, dopodiché è possibile lanciarlo direttamente o effettuare modifiche per poi fare `commit` su un branch di PRP.

Se si desidera effettuare cambiamenti al repository Git o semplicemente continuare ad utilizzarlo come strumento di versionamento, occorre creare un account GitHub e richiedere al fondatore (reperibile all'indirizzo mail jacopo.freddi@studenti.unipr.it) l'accesso al repository specificando il proprio username GitHub.
Se si desidera semplicemente testare il software, nel pacchetto definitivo è reso disponibile un file jar eseguibile che permette di avviare l'applicazione senza la necessità di replicare il repository.

5 La struttura software

La scelta di un linguaggio di programmazione fortemente Object-Oriented come Java ha permesso di modularizzare il codice in modo molto naturale. Di seguito vengono descritte le principali classi utilizzate (vengono tralasciate le interfacce e le classi di utilità, poco interessanti a questo fine).

5.1 PRPClient

PRPclient è la classe contenente il metodo principale. All'avvio vengono istanziate le classi di:

- gestione della tabella;
- gestione delle connessioni;
- interfacciamento tra lato client e lato server;
- gestione dei comandi da parte dell'utente;
- gestione della connessione con il proprio padre;

5.2 UserInformations

Tipo di dato POJO (Plain Old Java Object, oggetto il cui stato viene scritto alla costruzione e si mantiene immutato) che raccoglie le informazioni su nickname, indirizzo IP e porta di ascolto di un altro nodo.

5.3 TableManager

Classe che gestisce la tabella di inoltro del nodo: le entrate sono semplici coppie di nickname nella forma:

$\langle$`nickname del nodo A, nickname del nodo tramite il quale raggiungere A`$\rangle$.

Questa classe fornisce informazioni sulla struttura della rete, ma dev'essere aggiornata ogni volta che un cambio di topologia viene registrato.

5.4 ServerComponent

Lato 'server' dell'applicazione: riceve messaggi dal lato client e si preoccupa dell'invio e della ricezione dei messaggi verso e dalla rete. Incapsula le classi di:

- gestione della tabella;
- gestione delle connessioni;
- gestione della comunicazione con il client;
- una socket di tipo ServerSocket, atta a ricevere le comunicazioni dal client;

La scelta di un'architettura di questo tipo permette la massima libertà nella distribuzione delle classi: virtualmente, lato client e server del nodo potrebbero essere separati in due processi diversi, o addirittura due macchine diverse, rendendo il metodo di fruizione dell'applicazione molto più flessibile alle diverse esigenze.

5.5 ServerComponent

Gestisce i messaggi in arrivo dai nodi agendo di conseguenza.
Ogni modulo che comunica con un altro nodo, una volta ricevuto un messaggio, comunica a ServerComponent quale messaggio è stato ricevuto: ServerComponent si occupa quindi di propagare l'informazione a chi di dovere, ad esempio chiedendo a TableManager di aggiornare un'entrata.

5.6 POJOMessage

Tipo di dato POJO contenente i campi tipici di un messaggio.

5.7 ParentsManager

Modulo che mantiene la connessione del nodo con il proprio padre. Dato che l'algoritmo di selezione del nodo di backup implementato si applica solo in caso di disconnessione del proprio padre, solo questo modulo contiene la procedura di riconnessione al nodo di backup.

5.8 ParentClientManager

Modulo interno a ParentsManager, che gestisce direttamente la comunicazione con il nodo padre.

5.9 NetworkConnectionsManager

In apparenza un duplicato di TableManager, NetworkConnectionsManager gestisce una mappa $\langle$`nickname, informazioni`$\rangle$ da cui è possibile ricavare IP e porta di ascolto di un nodo a partire dal suo nickname.

5.10 ClientManager

Modulo di comunicazione con un generico nodo. In questa versione dell'algoritmo questo modulo viene utilizzato con tutti i nodi che richiedono una connessione (i figli). È perciò privo della procedura di riconnessione al nodo di backup.

5.11 ClientCommunicationManager

Modulo di comunicazione a basso livello: gestisce un thread separato per l'ascolto di messaggi in arrivo dal nodo ed invia i messaggi richiesti al nodo. Per ogni connessione nella rete viene creato un ClientCommunicationManager con relativo thread.

5.12 UserInterface

Modulo che si occupa del settaggio manuale del proprio indirizzo IP, porta di ascolto e nickname. Assolve inoltre alle funzioni di ascolto e gestione dei comandi da tastiera dell'utente.

6 Suggerimenti ed estensioni

La presente relazione ha delineato i requisiti e le linee guida per l'implementazione di un protocollo che soddisfi alle specifiche. Alcune possibili estensioni e miglioramenti sono elencati sotto:

1. aggiungere un meccanismo di cache che salva in locale i messaggi che dovrebbero essere inviati ai nodi disconnessi e cerca di inviarli a questi una volta che sono nuovamente raggiungibili, nell'ordine in cui sono stati inviati;
2. implementare una selezione user-friendly del nodo al quale connettersi;
3. implementare la rete come n-albero bilanciato invece che come grafo; rendere disponibili le procedure di disconnessione e riconnessione necessarie per mantenere bilanciato l'albero (seguendo le linee guida suggerite nel paragrafo apposito;
4. implementare la possibilità, da parte degli utenti, di definire gruppi di nodi per sfruttare al meglio il multicast;
5. implementare un meccanismo di 'friendship' tra nodi per definire i limiti di connettività e di invio dei messaggi;
6. implementare una versione del protocollo che possa funzionare sulla piattaforma Android: per come è stato realizzato il codice di esempio la parte di gestione della rete può essere riutilizzata, mentre l'interfaccia grafica andrebbe sviluppata. Date le risorse limitate a disposizione dei cellulari, è consigliato implementare la struttura ad albero bilanciato per minimizzare i costi in risorse della gestione della rete.
7. implementare una procedura di controllo di unicità del nickname dei nodi: alla connessione di un nodo, la validità del nickname dichiarato viene controllata nella rete e, se nessun nodo ha ancora richiesto tale nickname, questo viene prenotato ed acquisito dal nuovo nodo.
   In modo alternativo, assegnare un ID ad ogni nodo creato come funzione del nickname dichiarato e del timestamp di creazione. Notare che questa procedura è prona ad ambiguità nel caso un messaggio debba essere inviato ad un nickname posseduto da più di un nodo.
8. migliorare il punto precedente aggiungendo un metodo di autenticazione a chiave asimmetrica: Ogni nodo dovrà salvare in locale le chiavi pubbliche di ogni altro nodo;
9. aggiungere il timestamp ai campi dei messaggi e gestirne l'utilizzo;
10. per migliorare l'esperienza di utilizzo sul server, dividere il programma in due sezioni distinte: la parte server viene lanciata per prima e fornisce connettività verso la rete fungendo da nodo effettivo, mentre la parte client viene lanciata per seconda e si connette unicamente alla parte server. La parte server riceve messaggi dalla rete stampandoli e gestisce i messaggi provenienti dal client inoltrandoli sull'interfaccia giusta. Questa struttura permette di lavorare su due terminali separati e pulire l'interfaccia di invio dei messaggi dal flusso di messaggi ricevuti.