Protocollo PING/PONG - Esempio di implementazione in Java

Nicola Corti - 454413
Corso di Laurea Magistrale in Informatica - Università di pisa

22 Dicembre 2013

Sommario

Questa breve relazione ha lo scopo di illustrare i dettagli relativi alla realizzazione e all'uso di PingPong: un software che simula il comportamento del protocollo Ping/Pong di Gnutella, con o senza il meccanismo di *pong caching*.

Indice

Introduzione

1 Design & Implementazione

1.1 Classe per la gestione dei Messaggi

La prima classe che occorre descrivere è senza dubbio la classe Message, questa classe rappresenta infatti i messaggi che vengono scambiati fra i peer.

Si è cercato di realizzare la struttura struttura dei messaggi in modo che fosse quanto più fedele alla struttura definita nella RFC di Gnutella. Sono stati quindi definiti i seguenti membri:

funcion Un campo di tipo MessType (un enum) che può assumere valore PONG o PING e che rappresenta il tipo del messaggio,

GUID L'identificatore univoco del messaggio, generato in modo da concatenare il nome del peer che ha generato il messaggio al timestamp di generazione del messaggio,

HOP Numero di hop che ha effettuato il messaggio,

TTL *Time-to-leave* ovvero il numero di hop che può ancora effettuare il messaggio prima di essere scartato dalla rete,

sender Peer che ha generato il messaggio,

lastpeer Ultimo peer che ha effettuato il routing del messaggio,

sharedbytes Numero di byte condivisi dal peer che ha generato il messaggio,

sharedfiles Numero di file condivisi dal peer che ha generato il messaggio.

La classe offre inoltre alcuni costruttori per generare nuovi messaggi e per impostare correttamente tutti i campi. È inoltre presente un *copy constructor* (Message(Message m)) che crea una copia di un messaggio, utile quando un peer deve inviare un messaggio a tanti peer differenti, in modo da evitare conflitti sulle modifiche delle informazioni del messaggio (lastpeer, HOP, etc...).

1.2 Classi per la realizzazione dei Peer

Per realizzare il software è stata definita una gerarchia di classi che permettono di rappresentare il comportamento di ogni tipo di Peer. La gerarchia delle classi è rappresentata nel diagramma delle classi UML in figura 1

La radice della gerarchia è rappresentata dalla classe Peer, la classe in questione è astratta e contiene dei metodi che devono essere implementati dalle sottoclassi. Le varie sottoclassi della gerarchia contengono il codice per realizzare la logica dietro ogni tipo di peer. Le varie classi verranno presentate in dettaglio nei paragrafi seguenti.

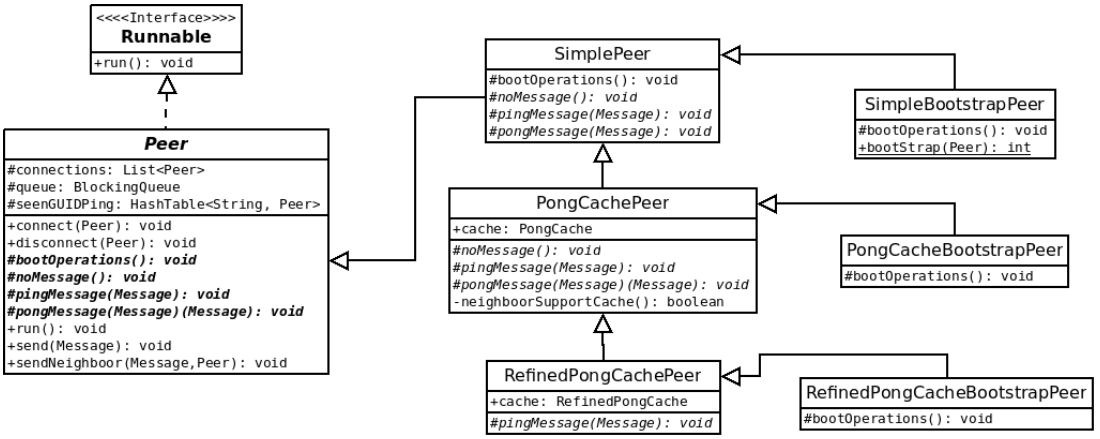

Figura 1: Gerarchia delle classi di Peer

1.2.1 Peer

La classe Peer contiene i metodi utili ad ogni tipo di peer. In particolare contiene i seguenti membri:

address Una stringa che rappresenta l'indirizzo del peer,

connection Una lista di oggetti di tipo Peer che rappresenta la lista delle connessioni del peer,

queue Una BlockingQueue che viene utilizzata per gestire la coda dei messaggi in entrata ad ogni peer. L'oggetto è *syncronized* e non crea problemi per quanto riguarda la concorrenza, offre inoltre metodi che sono bloccanti nel caso in cui non vi siano messaggi nella coda e risulta quindi l'ideale per modellare il comportamento dei peer,

seenGUIDPing Una HastTable per salvare i messaggi di Ping che sono passati, in coppia con il peer che li ha inviati, in modo da poter, realizzare il meccanismo del *backward routing*.

PING_COUNTER Un contatore per memorizzare il numero di Ping che sono passati,

PONG_COUNTER Un contatore per memorizzare il numero di Pong che sono passati.

Contiene inoltre i seguenti metodi di utilità per ogni tipo di peer:

connect Per connettere due peer fra di loro, le connessioni sono bidirezionali ed è sufficiente invocare connect sul peer A verso il peer B per connettere B verso A a sua volta,

disconnect Per disconnettere due peer fra di loro,

isconnected Per controllare se due peer sono connessi fra di loro,

send Per inviare un messaggio al peer su cui viene invocato (il messaggio viene accodato nella BlockingQueue),

sendNeighboor Per inviare un messaggio a tutti i vicini del peer su cui viene invocato.

Inoltre la classe Peer contiene il metodo run[1], che risulta essere un metodo fondamentale per l'esecuzione del comportamento dei peer.

Il metodo run contiene lo "scheletro" dell'esecuzione dei peer: il peer si avvia, esegue la funzione bootOperations(); dopo si mette in attesa di un messaggio sulla coda, se non riceve un messaggio entro un certo lasso di tempo esegue la funzione noMessage(), se riceve un messaggio di ping esegue pingMessage(), se invece riceve un messaggio di pong esegue pongMessage().

Ovviamente le funzioni bootOperations(), noMessage(), pingMessage() e pongMessage() sono funzioni astratte, devono essere implementate dalle sottoclassi.

Con una implementazione di questo genere si svincolano le sottoclassi da dover realizzare tutte le operazioni collegate alla BlockingQueue, in modo da dover implementare solamente la "business logic" del peer, ovvero le operazioni legate alla ricezione di messaggi di ping, pong, etc...

1.2.2 SimplePeer

La classe SimplePeer rappresenta un'implementazione di Peer che non utilizza il *pong caching*.

La classe SimplePeer implementa i metodi seguenti:

bootOperations Non effettuando nulla,

noMessage Non effettuando nulla,

pingMessage Salvando nel seenGUIDPing il ping che è stato ricevuto, propagandolo a tutti i vicini e generando un messaggio di Pong come risposta,

pongMessage Cercando nel seenGUIDPing il peer che aveva inviato il ping in passato, in modo da potergli inviare il pong ricevuto (cosidetto meccanismo di *backward routing*).

1.2.3 SimpleBootstrapPeer

La classe SimpleBootstrapPeer rappresenta una sottoclasse di SimplePeer che si comporta in modo analogo, tranne per il fatto che esegue la fase di bootstrap.

La classe SimpleBootstrapPeer implementa i metodi seguenti:

bootOperations Invocando la funzione statica bootStrap,

bootStrap Funzione statica per effettuare il bootstrap di un nodo: genera un messaggio di Ping e lo invia al primo peer nella lista dei peer connessi, attende dei pong di risposta per un certo lasso di tempo ed effettua la connessione con i peer in questione.

1.2.4 PongCachePeer

La classe PongCachePeer rappresenta una sottoclasse di SimplePeer e si tratta di un peer che utilizza il meccanismo di *pong caching* semplice.

La classe ha un membro cache di tipo PongCache per gestire la cache dei messaggi di pong ricevuti.

La classe PongCachePeer implementa i metodi seguenti:

[1] Implementato dall'interfaccia Runnable

pingMessage Controlla se ci sono pong a sufficienza nella cache per effettuare una risposta tramite la cache, se sì risponde con i pong della cache, altrimenti si comporta come pingMessage della superclasse (SimplePeer).

pongMessage Salva il pong nella cache e si comporta come pongMessage della superclasse (SimplePeer).

noMessage Invia un messaggio di Ping a tutti i vicini in modo da poter fare il *refresh* delle cache.

neighboorSupportCache Per controllare se i Ping vicini supportano il *pong caching*. Se infatti i vicini non supportassero il *pong caching* la frequenza con cui vengono mandati i messaggi di ping viene ridotta.

1.2.5 PongCacheBootstrapPeer

La classe PongCacheBootstrapPeer rappresenta una sottoclasse di PongCachePeer che si comporta in modo analogo, tranne per il fatto che esegue la fase di bootstrap.

La classe PongCacheBootstrapPeer implementa il metodo bootOperations invocando la funzione statica bootStrap della classe SimpleBootstrapPeer.

Si assume infatti che la fase di bootstrap sia analoga per tutti i tipi di peer, appena terminata la fase di bootstrap i peer si comportano come i peer della superclasse (in questo caso PongCacheBootstrapPeer si comporterà come un PongCachePeer appena terminata la fase di bootstrap).

1.2.6 RefinedPongCachePeer

La classe RefinedPongCachePeer rappresenta una sottoclasse di PongCachePeer e si tratta di un peer che utilizza il meccanismo di *refined pong caching*.

La classe è simile alla classe PongCachePeer tranne per il fatto che il membro cache è di tipo RefinedPongCache.

La classe RefinedPongCache implementa il metodo pingMessage rimuovendo dalla cache i ping scaduti e comportandosi come pingMessage della superclasse.

1.2.7 RefinedPongCacheBootstrapPeer

La classe RefinedPongCacheBootstrapPeer rappresenta una sottoclasse di RefinedPongCachePeer.

Per questa classe valgono i discorsi analoghi fatti per la classe PongCacheBootstrapPeer

1.3 Classi per la gestione della *pong cache*

Per supportare il meccanismo di *pong caching* è stato necessario definire due classi: PongCache e RefinedPongCache che gestiscono la cache dei pong nel caso semplice e *refined*.

In particolare la classe PongCache salva i pong all'interno di una HashTable ed offre dei semplici metodi per aggiungere e rimuovere pong:

addEntry Per aggiungere un messaggio alla cache (nel caso in cui fosse già presente un pong più vecchio proveniente dallo stesso peer questo viene aggiornato),

containsEnough Per controllare se la cache contiene un numero di pong sufficiente a permettere di rispondere senza trasmettere il ping,

getPongs Ritorna una lista di pong da poter mandare come risposta al ping ricevuto come parametro,

removeAll Per rimuovere tutti i pong dalla cache.

La classe RefinedPongCache aggiunge il metodo removeExpired per rimuovere i pong che sono scaduti.

1.4 Classe per la gestione della rete dei Peer

La classe che si occupa di gestire la rete dei peer è la rete PingPongNet. In particolare si è deciso di implementare i peer utilizzando i Thread di Java (si nota infatti che tutte le classi implementano l'interfaccia Runnable).

La classe in particolare si occupa di aggiungere peer di tutti i tipi alla rete, di connetterli, di generare il relativo Thread e di conservarlo in una lista. Una volta invocata la funzione start tutti i thread nella lista vengono avviati.

La classe si occupa anche di effettuare la stampa delle statistiche dei Peer leggendo i contatori PING_COUNTER e PONG_COUNTER dai Peer al fine di poter apprezzare le differenze fra le performance delle varie implementazioni.

1.5 Parametri del Sistema

Il sistema necessita di una serie di parametri che influenzano fortemente il comportamento del sistema. Tutti i parametri sono definiti come membri statici della classe Par e sono presentati nella tabella 1.

1.6 Altre Classi

Fra le altre classi è presente la classe Main che si occupa dell'esecuzione della simulazione. In particolare la classe si aspetta 3 file in input: un file contenente i peer, uno contenente le connessioni ed uno contenente la dinamica del sistema.

1.6.1 File dei peer

Il file dei peer deve essere un semplice file di testo contenente i peer del sistema, uno per riga terminati da un punto e virgola (;), ad esempio:

```
p1;
p2;
p3;
```

1.6.2 File delle connessioni

Il file delle connessioni deve essere un semplice file di testo contenente le connessioni fra i peer del sistema, una per riga, indicando i due peer separati da un punto e virgola (;), ad esempio:

```
p1;p3
p2;p3
p2;p1
```

Nome	Descrizione
MAIN_POLL _STATS_SECONDS	Secondi ogni quanto si desidera che il thread Main stampi le statistiche sul sistema
MESSAGE _MAX_TTL	Valore massimo del TTL del messaggio. Abbassando il valore si dimunisce il numero di peer che un messaggio può raggiungere. Il valore deve essere sempre >= 2
PEER_POLL _SECONDS	Numero di secondi che un peer deve stare in attesa di un messaggio, dopo questi secondo il peer esegue la funzione noMessage se non ha ricevuto messaggi
BOOTSTRAP _NEIGHBOOR _PONG_PEER	Parametro utilizzato dai peer che effettuano il bootstrap, rappresenta il numero di peer che si spera di riuscire a contattare nella fase di bootstrap
BOOTSTRAP _NEIGHBOOR _PONG_SECONDS	Numero di secondi che il peer in fase di bootstrap aspetta durante la ricezione dei Pong. Allo scadere di questi secondi il peer ha terminato la fase di bootstrap
PONG_CACHE _REQUIRED_PONGS	Numero minimo di Pong che sono richiesti nella cache per poter rispondere utilizzandola
PONG_CACHE _REFRESH _NOSUPPORT	Numero di secondi che vengono attesi per effettuare il refresh della pong cache nel caso in cui un vicino non supporti il *pong caching*
REFINED_PONG _CACHE_EXPIRING _SECONDS	Numero di secondi dopo i quali un Pong nella Pong Cache Refined scade

Tabella 1: Parametri del sistema con descrizione

1.6.3 File della dinamica

Il file delle dinamica contiene la dinamica del sistema, ovvero nuovi peer che si connettono, intervallati di pause. È possibile indicare un nuovo nodo che si connette ad un peer già esistente ed esegue la fase di bootstrap semplicemente indicando il nome del nuovo peer separato da un punto e virgola (;) dal peer a cui deve connettersi. È inoltre possibile indicare delle pause inserendo la parola `sleep` separata da un punto e virgola (;) dai secondi di pausa che si devono effettuare, ad esempio:

```
n1;p1
n2;p3
sleep;5
n3;p1
```

Il nome dei nuovi peer **non** deve essere indicato all'interno del file dei peer, e al contempo non deve essere uguale al nome di uno dei peer contenuto nel file dei peer.

1.6.4 Generazione della rete

È possibile generare una nuova rete utilizzando la classe GraphGenerator che si aspetta in input due parametri:

- Il numero di nodi della rete,
- La probabilità (da 0 a 1) che sia presente una connessione fra due peer.

In particolare valori di probabilità più bassi permettono di creare reti più sparse, mentre valori più alti generano reti più dense fino a generare un *mesh* per valori di probabilità pari ad 1.

Si consiglia di utilizzare valori di probabilità bassi, in quanto la rete Gnutella è una rete molto sparsa. Si consiglia inoltre di non utilizzare reti di dimensioni molto grosse, in quanto i peer sono implementati tramite i Thread e se si utilizza una macchina che supporta un numero ridotto di thread (2, 4 o 8 thread) in esecuzione parallela il comportamento può discostarsi molto da quello previsto.

2 Performance

Al fine di valutare le performance delle varie implementazioni sono state realizzate delle piccole analisi utilizzando i contatori PING_COUNTER e PONG_COUNTER di ogni peer. In particolare la classe PingPongNet ci permette di interrogare tutti i peer e calcola i valori aggregati di tutti i contatori.

2.1 Numero dei messaggi per ogni peer

Nei grafici seguenti si possono vedere come si evolve il totale dei messaggi di Ping/Pong per ogni peer appartenente ad una rete di 15 peer (generata con grado di probabilità 0.1) a cui si connettono 6 nuovi peer (da n1 a n6).

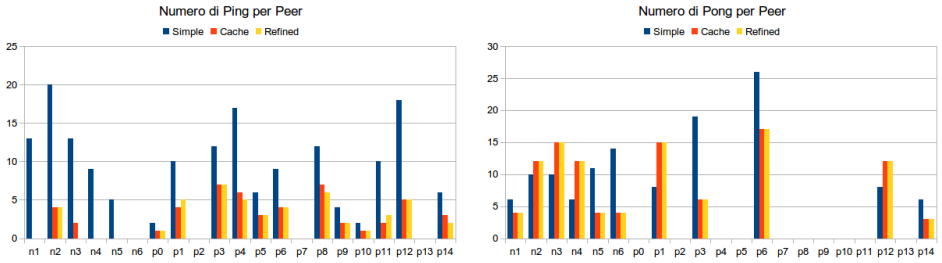

Figura 2: Grafici che mostrano il numero di ping e di pong che sono stati ricevuti da ogni peer della rete

Si può notare come il numero di ping e di pong sia molto ridotto nel caso in cui si utilizzi il pong caching. Dai grafici si nota chiaramente che i peer che sono stati utilizzati per fare il bootstrap siano p1, p3 e p6 dato che hanno un numero di pong ricevuto molto elevato.

Si noti invece come alcuni peer non sono stati mai raggiunti da alcuni messaggi, questo è possibile in quanto la topologia della rete è molto sparsa e i test sono stati effettuati con un TTL basso, per cui molti messaggi vengono scartati prima di raggiungere i luoghi più lontani della rete.

Per il peer p1 notiamo però che il numero di pong è aumentato nel caso del pong caching, ciò può essere dovuto al fatto che il peer non è riuscito ad avere un numero di pong sufficienti a rispondere tramite la cache; ha quindi dovuto girare il messaggio di ping ed ha inoltre contattato i vicini per ricevere altri pong con cui riempire la cache.

2.2 Numero di messagi totali

Nei seguenti grafici si nota maggiormente il miglioramento delle performance indotto dal pong caching, in particolare sono stati effettuati test su reti da 10, 30 e 50 peer (sempre

generate con probabilità pari a 0.1) e si può notare chiaramente come vi sia una netta diminuzione del numero dei messaggi ricevuti.

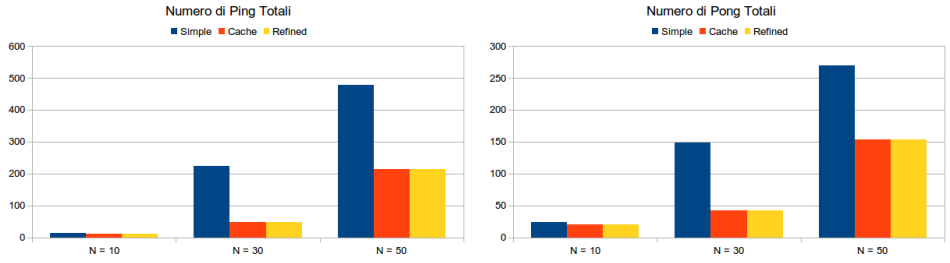

Figura 3: Grafici che mostrano il numero di ping e di pong che sono stati ricevuti in totale da tutti i peer della rete

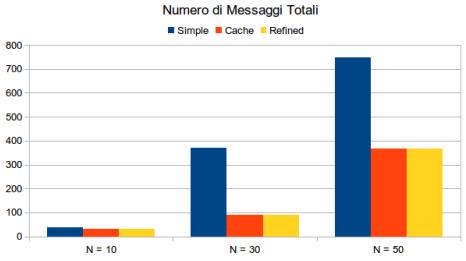

Figura 4: Grafico che mostra il numero totale di messaggi che sono stati ricevuti da tutti i peer della rete

Si noti come il caso refined abbia risultati identici al caso con la cache semplice, questo è dovuto al fatto che sono state effettuate le misurazioni dei messaggi al termine della fase di bootstrap. Dopo la fase di bootstrap la rete che non implementa il pong caching si stabilizza, mentre la rete con il pong caching semplice o refined continua ad inviare messaggi per effettuare il *refresh* della cache.

2.3 Messaggi legati al refresh

Nel seguente grafico si vede come si evolve il numero dei messaggi ricevuti nel caso con il *pong caching* semplice o refined:

Si può notare come il numero dei messaggi cresca nel tempo. Il fatto di usare un *pong caching* di tipo refined comporta un leggero aumento del numero dei messaggi, in quanto qualche pong memorizzato nella cache dei peer potrebbe scadere e portare i peer a non poter rispondere con i pong in cache (in quanto non risultano essere in numero sufficiente).

Questo fenomento si nota dal grafico in cui le linee continue (legate a casi refined) risultano avere valori leggermente maggiori.

Non si pensi però che il caso refined risulta peggiore, in quanto offre come risposte dei pong che riusciranno a far stabilire una connessione con probabilità maggiore rispetto ai pong del caso semplice. Purtroppo questo fatto non si evince da questo grafico, ma si dovrebbe implementare all'interno della rete il fenomeno della caduta dei peer o dell'abbandono della rete.

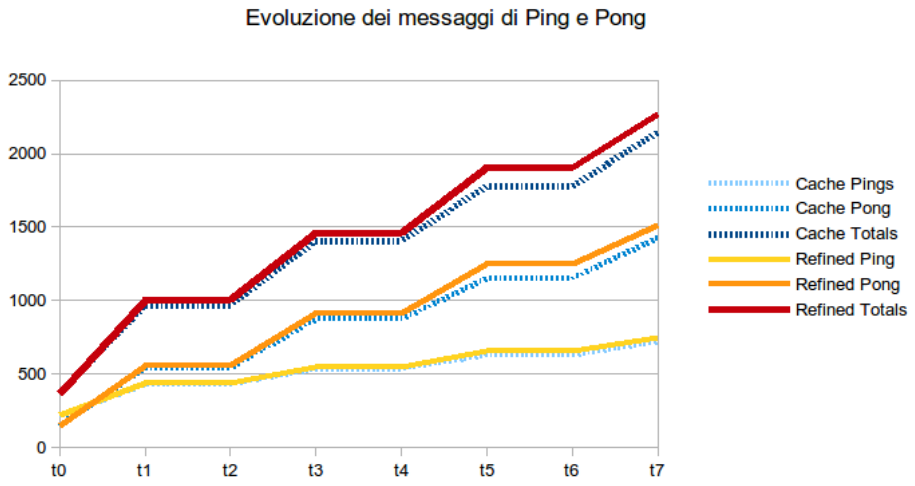

Figura 5: Grafico che mostra l'evoluzione del numero di messaggi nel tempo nella rete con *pong caching* semplice e refined

3 User Guide

Per installare il software è necessario posizionarsi all'interno della directory dove è contenuto il software ed invocare da terminale il comando

```
ant
```

Nel caso si fosse interessati a conoscere i dettagli del processo di installazione, procedere con la lettura, altrimenti procedere dalla sezione relativa all'utilizzo del software.

3.1 Installation

Il software è corredato di un file **ant** (il file `build.xml`) che offre dei target per automatizzare il processo di compilazione e di configurazione del software.

In particolare è possibile inizializzare l'ambiente di lavoro utilizzando il comando

```
ant init
```

che provvede a creare le cartelle necessarie per completare il processo di compilazione.

Per compilare il progetto è necessario eseguire il target

```
ant build
```

che provvederà ad invocare il compilatore `javac` per compilare i sorgenti presenti all'interno della cartella `src/`, i file `.class` generati si troveranno all'interno della cartella `bin/`. Il target `build` provvede ad invocare automaticamente il target `init`, per cui non è necessario invocare direttamente il target `init` a meno che non si sia interessati a configurare l'ambiente senza effettuare la compilazione.

Per pulire la cartella `bin/` al fine di avere un ambiente pulito per poter effettuare una nuova compilazione è possibile utilizzare il target

```
ant clean
```

È infine possibile generare un file `jar` contenente tutti i file compilati. Per farlo è sufficiente invocare il target

```
ant jar
```

Verrà generato un file chiamato `pingpong.jar` all'interno della cartella principale del software. Per avviare il file `jar` è necessario invocare il comando

```
java -jar pingpong.jar [parametri]
```

3.2 Documentation

Al fine di rendere il codice sorgente più comprensibile il software è stato corredato di documentazione. In particolare tutte le parti del codice che potrebbero risultare di difficile comprensione sono state commentate. Inoltre ogni funzione e classe del software è stata documentata con il formato javadoc, la documentazione generata può essere visionata all'interno della cartella doc/ e può essere rigenerata utilizzando il comando

```
ant javadoc
```

Per una comprensione organica del software si consiglia la lettura della seguente relazione nella sua interezza. La presente relazione viene rilasciata in Pdf ed in LaTeX e può essere ricompilata utilizzando il comando[2]

```
ant latex
```

3.3 Software Usage

Una volta compilato il software è possibile invocare il software tramite lo strumento `ant`, il software permette di essere invocato indicando alcuni parametri al fine di effettuare simulazioni con reti e protocolli differenti.

Il software può essere invocato tramite il comando

```
ant <mode> [-Dpeer=filepeer] [-Dconn=fileconn] [-Ddyna=filedyna]
```

I parametri sono i seguenti:

-Dpeer= Permette di impostare il file dei peer.

-Dconn= Permette di impostare il file delle connessioni.

-Ddyna Permette di impostare il file della dinamica.

Le modalità sono le seguenti:

Simple Esegue una simulazione in cui i peer della rete non implementano il *pong caching*,

Cache Esegue una simulazione in cui i peer della rete implementano il *pong caching* semplice.

Refined Esegue una simulazione in cui i peer della rete implementano il *refined pong caching*.

[2]Si noti che tale target funziona solamente se installato il software pdflatex e se eseguito in ambiente UNIX

Generazione della rete È inoltre possibile effettuare la generazione di una coppia di file da utilizzare per effettuare la simulazione. Per generali è sufficiente invocare il comando:

```
ant Generate [-Dn_peer=numberofpeer] [-Dt_conn=connectprob.ty]
```

Dove i parametri sono i seguenti:

-Dn_peer= Permette di impostare il numero dei peer che faranno parte della rete simulata,

-Dt_conn= Permette di impostare la probabilità che esista una connesione fra due generici peer.

4 Future Improvements

Sono possibili ulteriori miglioramenti al software che potrebbero essere implementati in versioni future:

- Utilizzare i campi sharedfiles e sharedbytes all'interno dei Message in modo che i peer che stanno effettuando il bootstrap possano effettuare una scelta più oculata dei peer a cui connettersi al fine di favorire i peer che stanno condividendo più files e di scoraggiare i peer che stanno condividendo poco.
- Aggiornare le classi Message e Peer per permettere la ricezione e l'invio di messaggi di QUERY, QUERY_HIT, etc...
- Aggiornare la funzione getPongs di PongCache in modo che si riceva un insieme di pong con valori di HOPS differenti, in modo da permettere al peer che si sta connettendo di connettersi a peer che sono sia vicini, che distanti.
- Implementare la possibilità che un nodo abbandoni la rete spontaneamente oppure non spontaneamente (per esempio in seguito ad un guasto della rete).
- Inserire un timer che risponda ad un ping solamente se non sono stati ricevuti ping sulla stessa connessione per un certo tempo (ad esempio 1 secondo)[3].
- Inserire una dimensione massima della cache, in modo da non sovraccaricare la memoria dei peer.

5 Licence

Per qualsiasi problema è possibile contattare lo sviluppatore all'indirizzo e-mail `cortin [at] cli.di.unipi.it`.

[3] Secondo quanto definito nell'RFC di Gnutella: `http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html`