

Relazione sul Progetto di Sistemi di Simulazione: Analisi della stabilità del protocollo di DHT Symphony

MATTEO BRUCATO e MIRO MANNINO, Università di Bologna

Symphony è un protocollo di overlay per sistemi peer-to-peer in grado di mantenere una tabella hash distribuita (DHT) attraverso nodi che risiedono in una rete geografica.

1. INTRODUZIONE

Symphony [?] è un protocollo per tabelle hash distribuite (DHT) che, attraverso una rete di overlay costruita in base a particolari distribuzioni armoniche, è in grado di garantire lookup efficienti, poli-logaritmici nel numero di hop (salti), ovvero nella sua *latenza*¹.

Attraverso un concetto denominato *greedy routing*, è stato mostrato come sia possibile indirizzare e consegnare un messaggio ad un qualsiasi nodo in una rete in al più $O(\log^2 n)$ salti (un fenomeno chiamato anche *Small World*) [?]. Symphony si basa esattamente su questi concetti, i quali vengono applicati magistralmente al caso di reti peer-to-peer e al problema di creare DHT. In particolare, Symphony è in grado di garantire una latenza di $O(\frac{1}{k} \log^2 n)$ salti, dove $k = O(1)$ è il numero di link che ogni nodo mantiene verso altri nodi della rete.

Nell'articolo in cui Symphony è stato presentato [?], vengono evidenziati i molteplici vantaggi del protocollo rispetto ai suoi predecessori (Chord, Viceroy, ecc.). Il vantaggio più evidente è quello di richiedere un numero costante $k = O(1)$ di link verso altri nodi della rete, a differenza di protocolli come Chord [?] che richiedono un numero logaritmico di link uscenti.

Symphony, inoltre, è dimostrato essere *scalabile* nella dimensione della rete,

Infine, gli autori dichiarano anche il sistema *stabile*, ovvero capace di funzionare nel caso in cui gli host si uniscano alla rete ed escano dalla rete in maniera del tutto arbitraria, tipicamente con brevi periodi di vita. Ciononostante, un'approfondita analisi dell'articolo mostra come gli esperimenti effettuati per stabilire la stabilità di Symphony offrano solamente una panoramica limitata sui possibili scenari che possano veramente essere fonte di instabilità. I test effettuati non sono tali da stressare il sistema fino ai punti più critici, in cui un'evidente instabilità possa essere misurata.

Lo scopo del presente elaborato è quello di studiare a fondo la stabilità del protocollo Symphony, e di stabilire in maniera più accurata se tale protocollo può essere veramente considerato stabile e in che misura. Più precisamente, ci poniamo l'obiettivo di valutare la stabilità del protocollo sotto carichi di stress causati da *churn*, ovvero da alte frequenze di entrata e uscita dei nodi della rete. Studieremo sotto quali condizioni di stress la stabilità del protocollo vale e in quali condizioni essa comincia a vacillare.

Attraverso una serie di esperimenti in simulazione, mostreremo come il protocollo Symphony risponda molto bene a situazioni di stress causato da *churn* simili a quelle testate dagli autori del protocollo. E' solo con l'ausilio di un *motore di churn* molto sofisticato che siamo in grado invece di stressare il sistema fino a fargli perdere la sua stabilità. I nostri esperimenti, infatti, mostrano che

¹In sintonia con la terminologia tipica della letteratura peer-to-peer, indichiamo col termine *latenza* il numero medio di hop per l'operazione di lookup.

La relazione è organizzata nel seguente modo. In Sezione 2 descriveremo brevemente il protocollo Symphony. In Sezione 3 descriveremo formalmente il concetto di stabilità che sarà oggetto di valutazione del protocollo in esame. In Sezione 4, presenteremo i dettagli sul simulatore realizzato, il livello di astrazione del simulatore stesso e dettagli sulla sua realizzazione. La Sezione 5 raccoglierà i risultati sperimentali delle simulazioni, mentre la Sezione 6 illustrerà le nostre conclusioni.

2. IL PROTOCOLLO SYMPHONY

Symphony è un protocollo DHT basato su un'overlay strutturato con topologia ad anello. Gli identificatori dei nodi sono selezionati in maniera randomica uniforme nell'intervallo $[0, 1)$. Ogni nodo p_i con identificatore x gestisce il sotto-intervallo dell'anello $(y, x]$ se y è l'identificatore del peer p_{i-1} immediatamente precedente al peer p_i . In questo modo ogni valore $x \in [0, 1)$ è gestito (univocamente) da uno ed un solo peer, che chiameremo il *manager* di x nel resto della presente relazione.

2.1 Collegamenti tra nodi

I nodi della rete Symphony sono collegati tra loro secondo due tipi di link: *short link* e *long link*, ovvero collegamenti a breve e lungo raggio. I collegamenti short servono a creare la struttura ad anello vera e propria: un peer p_i è collegato per mezzo di due short link ai due peer p_{i-1} e p_{i+1} (supponendo sempre aritmetica modulo n), ovvero il suo predecessore e il suo successore nell'anello. I collegamenti long, invece, servono a permettere salti a lunga distanza nell'anello, i quali determinano la complessità logaritmica del protocollo di routing di Symphony. I long link sono quindi la caratteristica più importante del protocollo in esame.

Ogni nodo mantiene $k \geq 1$ long link. La scelta dei peer a cui collegarsi è la peculiarità principale del protocollo Symphony. Ogni nodo, quando deve decidere a chi collegarsi per mezzo di un long link, utilizza un numero randomico $x \in [0, 1)$ selezionato sulla base di una distribuzione di probabilità appartenente ad una famiglia di distribuzione armoniche, una versione continua della distribuzione discreta di Kleinberg.

2.2 Join

Quando un peer vuole entrare nella rete

3. ANALISI DELLA STABILITÀ

Nell'articolo originale [?], gli autori Manku et al. testano la stabilità del protocollo Symphony in una cosiddetta *rete dinamica*, ovvero una rete in cui i nodi entrano ed escono arbitrariamente. In particolare, gli autori studiano una rete di centomila nodi, ognuno dei quali ha un numero logaritmico di vicini e alterna uno stato di attività ad uno stato di inattività. I tempi di attività e inattività sono tratti da due distribuzioni esponenziali diverse, tali che i nodi restino attivi per poco tempo (in media mezz'ora) e inattivi per il resto della giornata (23 ore e mezza in media). Le distribuzioni esponenziali fanno sì che la stragrande maggioranza dei nodi rispecchino questo comportamento.

Questo setup rispecchia certamente alcuni scenari veritieri di utilizzo dei sistemi DHT. I nodi entrano nella rete una volta al giorno e vi rimangono per pochissimo tempo (forse il tempo necessario per fare una semplice ricerca e scaricare uno o due file). Ciononostante non è certamente atto a coprire la maggioranza dei casi di utilizzo possibili, specialmente quelli in cui la rete venga stressata in maniera molto più pesante. Si pensi ad esempio ad un utilizzo di una DHT da parte di altri sistemi, in sistemi distribuiti integrati fra loro. In tali situazioni, non si può fare nessuna supposizione né sugli orari di utilizzo, né sulla durata d'utilizzo o sulla frequenza di entrata nella rete.

Non solo lo scenario usato nei test dagli autori di Symphony prevede una frequenza di entrata e uscita piuttosto basse, ma gli autori assumono anche una crescita della rete molto controllata. Nei

loro test, infatti, essi fanno crescere la rete in maniera lineare durante l'arco di una giornata. Alla fine della prima giornata, tutti i centomila nodi sono entrati nella rete, e vi rimangono durante tutto l'arco della giornata successiva. Durante il terzo giorno, invece, i peer vengono fatti uscire dalla rete a intervalli regolari. Non siamo riusciti ad immaginare, in questo caso, un esempio di utilizzo reale che rispecchi questo particolare test. Ma soprattutto, ancora, questo test non è atto a stressare la rete con un'adeguata frequenza di churn, o in casi di altissima e imprevedibile attività.

Infine, i test riguardanti la stabilità presenti nell'articolo considerano solo reti senza re-linking. Reputiamo molto interessante stabilire cosa accade in una rete sotto churn nel caso in cui il re-linking sia attivo, in quanto esso forza molti peer a distruggere e ricreare tutti i propri long link, i quali sono quelli che contribuiscono maggiormente alle performance del protocollo stesso.

3.1 Modello di stabilità

Il modello di stabilità che vogliamo considerare nel presente elaborato è più complesso e a grana più fine di quello utilizzato nel paper originale. Innanzitutto, osserviamo come il tempo di permanenza di un certo peer nella rete è influente rispetto la misura di stabilità della rete. Sapere che un peer sta nella rete per mezz'ora o per un giorno non fa alcuna differenza. Ciò che può maggiormente influire sulla stabilità è il fatto che una grande quantità di peer chiedano un lookup mentre la rete è in fase di creazione a seguito di *join* (entrate) e *leave* (uscite) dei peer.

L'ipotesi alla base dei nostri test è che durante i join (e/o i leave) l'anello di Symphony possa avere potenziali punti di inefficienza dati dal fatto che i peer che stanno entrando non hanno ancora terminato la fase di linking necessaria alla rete per funzionare nei tempi logaritmici. Aumentare il numero di peer che fanno join o leave per istante di tempo dovrebbe quindi evidenziare tale inefficienza nelle misurazioni delle latenze dei lookup.

Ciò che quindi sembra poter avere un impatto più preponderante sulla stabilità non è il tempo di permanenza del singolo peer, ma la *frequenza del fenomeno di churn* (sia esso di leave o di join). Il nostro modello, quindi, prevede di potere valutare la stabilità del sistema al variare della frequenza di join e/o di leave nel sistema.

Questo approccio offre anche numerosi vantaggi in termini di flessibilità nei test. [poter concentrarci sulla frequenza di churn ignorando dettagli come tempo di permanenza, orario di entrata e uscita, effettiva grandezza della rete ad ogni istante di tempo permettono di effettuare test di stabilità più generali]

3.2 Misurare la stabilità

Una buona rete peer-to-peer senza struttura garantisce in generale che un lookup non richieda un numero di salti maggiore del numero di nodi della rete. Nel caso pessimo, infatti, la rete invia il messaggio di lookup a tutti i peer della rete finché il peer destinatario non viene contattato. L'upper bound del lookup è cioè $O(n)$ per qualunque rete peer-to-peer non banale.

Una rete strutturata come Symphony, in generale, garantisce molto meglio: un upper bound logaritmico nel numero di salti necessari ad un singolo lookup. Ciononostante, se n peer dovessero entrare tutti contemporaneamente, prima che la rete sia riuscita a creare la struttura necessaria a garantire l'upper bound teorico, le prestazioni potrebbero essere molto peggiori del previsto, fino al caso pessimo di lookup in $O(n)$ salti, come nel caso di rete senza struttura.

Bisogna inoltre considerare che l'upper bound teorico non corrisponde per forza al numero di hop necessari, e può essere anche molto più alto (pessimistico) rispetto a ciò che avviene nella realtà. Una buona struttura e un buon protocollo possono, in media, garantire numero di salti molto minori dell'upper bound teorico.

Ciononostante, è plausibile aspettarsi che uno stress maggiore o minore del sistema possa modifica-

re, in media, il numero effettivo di salti necessari al processamento dei lookup. Ciò potrebbe portare le prestazioni del sistema ad essere nella media più vicine all'upper bound teorico. Nel seguito ci riferiamo a questo trend, se presente e misurabile, come una forma di *instabilità* di un protocollo di lookup. Di contro, un sistema è più stabile tanto più le sue prestazioni rimangono eguali indipendentemente dallo stress momentaneo subito dal sistema, e tanto più il numero di salti dell'algoritmo di lookup misurati sperimentalmente si discosti dall'upper bound teorico. Tale stress è caratterizzato, come detto in precedenza, dalla frequenza di join o di leave dei peer della rete.

Introduciamo quindi, ai fini della nostra analisi, le seguenti definizioni di *stabilità*: (1) la stabilità di un singolo lookup, (2) la stabilità di una serie di misurazioni di lookup, e infine la (3) stabilità di un protocollo di DHT.

Definition 3.1. (STABILITÀ DI UN LOOKUP) Dato un certo protocollo DHT e una misurazione relativa ad un singolo lookup (h, n) dove h è il numero di hop (salti) del lookup e n è il numero di peer presenti nella rete all'istante della terminazione del lookup, la stabilità del lookup è definita come:

$$s(h, n) = 1 - \frac{h}{n}$$

L'instabilità di un singolo lookup è data da $\frac{h}{n}$ se h è il numero di hop del lookup e n il numero di nodi al momento presenti nella rete. Infatti, più h si avvicina ad n , più tale numero si avvicina ad 1, più significa che il numero effettivo di hop si avvicina all'upper bound teorico e quindi alla nostra idea intuitiva di instabilità. La stabilità, così come appena definita, quindi è data dall'opposto della instabilità, ovvero $1 - \frac{h}{n}$.

La stabilità di una singola misurazione è zero nel caso in cui $h = n$, ovvero nel caso di massima instabilità, data da una rete in cui il protocollo DHT ha perso la sua capacità di garantire un lookup logaritmico in quanto l'effetto del churn ne ha distrutto le proprietà fondamentali. La stabilità non può mai essere minore di zero, in quanto $O(n)$ è l'upper bound dei singoli lookup, come descritto in precedenza, ovvero $h \leq n$. Essa è uguale ad 1 nel caso in cui il numero di hop effettivi sia zero (ad esempio, il file ricercato si trova nel nodo stesso che lo sta cercando). Non può essere maggiore di 1 poiché sia h che n sono sempre numeri positivi. Pertanto, vale il seguente lemma:

LEMMA 3.2. CARATTERIZZAZIONE DELLA STABILITÀ

$$0 \leq s(h, n) \leq 1$$

Dato un insieme di misurazioni su diversi lookup, definiamo la stabilità delle misurazioni come la media delle stabilità misurate:

Definition 3.3. (STABILITÀ DI UN INSIEME DI MISURAZIONI) Dato un certo protocollo DHT e un insieme di r misurazioni della funzionalità di lookup $H = \{h_1, h_2, \dots, h_r\}$, dove h_i è il numero di hop (salti) relative all' i -esimo lookup, e dato un insieme di r misurazioni $N = \{n_1, n_2, \dots, n_r\}$, tale che n_i corrisponde al numero di peer presenti nella rete alla fine dell' i -esimo lookup, la *stabilità* del protocollo è definita come:

$$stability(H, N) = \frac{\sum_i s(h_i, n_i)}{r} = 1 - \frac{\sum_i h_i/n_i}{r}$$

Diciamo che un protocollo è stabile se la sua stabilità, così come appena definita, non varia di molto al variare delle condizioni di stress della rete causate dai fenomeni di churn più o meno frequenti. In altre parole, utilizziamo la seguente definizione:

Definition 3.4. (STABILITÀ DEL PROTOCOLLO) Sia dato un certo protocollo DHT e un insieme di R insiemi di misurazioni $M = \{(H_1, N_1), (H_2, N_2), \dots, (H_R, N_R)\}$ tale che ogni insieme di misurazioni

(H_i, N_i) sia stato effettuato ad un certo livello di stress della rete dato da una certa frequenza f_i di churn. Sia $S(M) = \{stability(H_1, N_1), stability(H_2, N_2), \dots, stability(H_R, N_R)\}$. Il protocollo è detto essere *stabile* sse

$$2 \cdot \text{Std}[S(M)] \leq \epsilon$$

per un certo $0 \leq \epsilon \leq 1$.

Si noti che, visto che la stabilità è un valore compreso tra 0 e 1, $\text{Std}[S(M)]$ può essere al massimo $1/2$, il che giustifica il fattore moltiplicativo 2 presente nella definizione. Intuitivamente, la presente definizione di stabilità è parametrica rispetto ad un requisito di stabilità ϵ . Se $\epsilon = 1$, qualsiasi protocollo è stabile, se $\epsilon = 0$ un protocollo è stabile solo se la stabilità è totalmente indipendente dalla frequenza dei churn, ovvero se il numero di hop rispetto alla grandezza della rete è sempre lo stesso indifferentemente da quanti nodi entrano o escono dalla rete per unità di tempo.

Il requisito di stabilità ϵ può essere settato liberamente a seconda dei requisiti di stabilità del protocollo, e dipende quindi maggiormente dal campo applicativo specifico della DHT. Se un protocollo è stabile per un certo ϵ' , allora è stabile per ogni $\epsilon'' \leq \epsilon'$. Un problema interessante in questo contesto è quindi stabilire ϵ^* , ovvero il minimo valore di ϵ per cui il protocollo è stabile. Nel paragonare la stabilità di due protocolli DHT, quindi, si possono utilizzare proprio i loro rispettivi valori di ϵ^* . Il protocollo con ϵ^* minore è più stabile in quanto la sua stabilità è meno dipendente dalla frequenza dei churn.

4. IL SIMULATORE E IL LIVELLO DI ASTRAZIONE

4.1 Strumenti utilizzati

Lo studio di simulazione è stato realizzato mediante l'uso del simulatore Omnet++ 4.2.2[?]. Quest'ultimo è una libreria (ed anche un framework) per creare simulazioni, utilizzando C++, che permette di eseguire simulazioni complesse in tempi ragionevoli. La struttura modulare di Omnet++ ha poi favorito la realizzazione della simulazione in esame, suddividendo le varie componenti in modo da poterle gestire limitandone la complessità. Le simulazioni sono state compilate ed eseguite in due diverse architetture: Linux e Mac OS X. Ciò ha favorito anche la stesura di un codice più robusto, in grado di gestire le diversità dei due sistemi operativi, che malgrado simili tra loro, riportano leggere differenze che evidenziano errori differenti.

Rispettando le convenzioni di Omnet++, il protocollo simulato è stato realizzato mediante l'uso di:

- File 'NED' per modellare le varie componenti, come la rete, i peer, i canali di comunicazione ed il churner.
- File 'msg' che definiscono i vari messaggi che i vari peer possono mandarsi.
- File sorgenti, contenenti le classi che implementano i vari moduli descritti nei file NED.
- File 'omnet.ini', che permette di configurare ogni run in modo parametrico.

Le analisi dei dati di ogni simulazione sono state poi effettuate sia con gli stessi strumenti messi a disposizione da Omnet++, che da piccoli script appositamente progettati per filtrare e modificare i dati grezzi, in modo da poterli poi rappresentare graficamente mediante l'uso di gnuplot.

Inoltre, le librerie grafiche messe a disposizione, hanno permesso di rappresentare in maniera graficamente efficace la rete di overlay, evidenziando i vari peer con i loro collegamenti, ed i vari messaggi scambiati. Ciò ha anche aiutato la fase di debug.

4.2 Simulazione

La simulazione è costituita da una rete, contenente una serie di peer collegati tra loro attraverso dei canali di comunicazione. Inoltre la simulazione ha un'entità con un comportamento diverso dal generico peer, chiamata churner, che stabilisce quando i vari peer devono entrare ed uscire dalla rete.

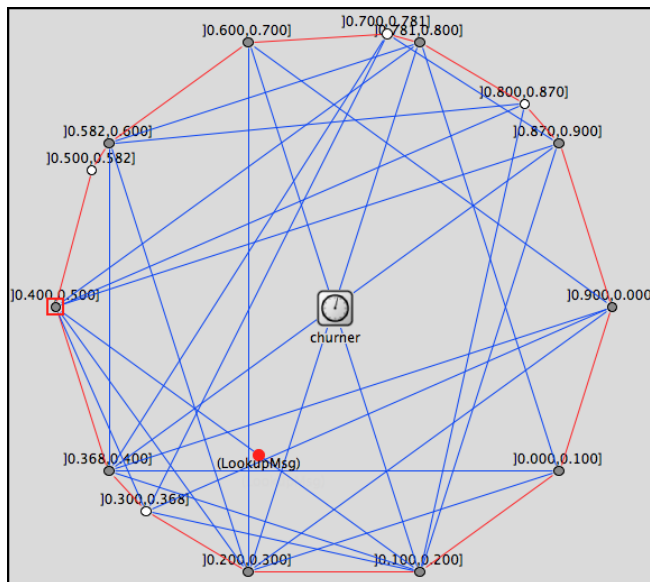


Fig. 1. Una screenshot della simulazione. Possiamo notare i long link colorati di blu ed i short link colorati di rosso. I nodi più scuri sono i peer statici, mentre quelli più chiari sono quelli dinamici. Ogni peer ha poi una label che indica il range di valori che vengono amministrati da tale nodo.

La rete funge da contenitore per tutta la simulazione. Questa stabilisce il numero di peer che devono essere presenti all'interno della simulazione. Inoltre, definisce i canali presenti nella rete, con un delay di 100ms e con un bandwidth di 10Mbps.

Il peer rappresenta invece un generico nodo della rete. Questa mantiene alcune informazioni necessarie per distinguersi:

- Un $id \in [0, 1]$, che identifica univocamente il peer all'interno della rete di overlay Symphony.
- Un canale di comunicazione verso il nodo precedente, ed un altro per il successivo. Questi costituiscono ciò che in Symphony vengono chiamati *short link*.
- Un array di canali di comunicazione, che comunicano con peer arbitrari. Questi costituiscono ciò che in Symphony vengono chiamati *long link*.

Inoltre ogni peer può contare su una serie di parametri che sono identici per ogni peer, che delineano in modo parametrico il loro comportamento finale.

- Il parametro k , che costituisce il numero di long link che il peer deve provare a fare.
- Un parametro per stabilire il numero di tentativi che il peer è disposto a fare per costruire i long link. Ciò viene accuratamente descritto nell'articolo originale di Symphony.
- Uso o meno del relinking.

Ogni peer poi può essere utilizzato, all'interno della simulazione, in due modi piuttosto differenti: in maniera statica ed in maniera dinamica.

4.2.0.1 Peer statico. Il peer statico viene inserito all'interno della rete di overlay (cioè di Symphony) in maniera automatica, prima che la simulazione abbia inizio. Tali nodi non possono uscire (e tantomeno entrare) da tale rete. Pertanto, gli short link vengono creati prima della partenza della simulazione,

utilizzando i costrutti dei file NED, messi a disposizione da Omnet++, collegando ogni coppia di nodi adiacenti con uno short link. Invece, per i long link, il potere espressivo troppo debole di tali costrutti non consente di crearli in modo corretto. È infatti possibile costruirne solamente una approssimazione, ma non si garantirebbe l'assenza di conflitti (e.g. uno short link collega due nodi già connessi), e tantomeno non si garantirebbe lo stesso procedimento descritto da Symphony per la loro creazione (e.g. massimo numero di tentativi per la loro creazione, che permette di risolvere quelle situazioni dove la rete è satura). Pertanto, i long link vengono creati dagli stessi peer statici all'inizio della simulazione, in maniera quasi istantanea (non vengono utilizzati i messaggi di lookup in questo caso, ogni peer sa esattamente chi sono i peer che amministrano un particolare id). Pertanto è necessario che le simulazioni attendano questo periodo transiente (che chiamiamo *periodo di warmup*). In particolare è il churner che deve attendere tale periodo prima di far entrare ed uscire i peer, allo stesso modo devono attendere anche le misurazioni effettuate.

Ogni peer statico p_i ha inizialmente un $id_i \in [0, 1[$ ben preciso: se rappresentiamo i peer statici come $\{sp_1, sp_2, \dots, sp_n\}$, abbiamo che $id_i = i/n$. Ciò significa che gli id dei peer statici sono uniformemente distribuiti all'interno dell'anello che rappresenta la rete di overlay di Symphony.

4.2.0.2 Peer dinamico. I peer dinamici vengono inseriti all'interno della rete che contiene tutti i peer della simulazione, ma non all'interno della rete di overlay. Pertanto inizialmente esso non ha nessun long link, e nessun short link.

Un generico peer dinamico rimane quindi in disparte fino a quando il churner, attraverso un messaggio, non gli chiede di entrare a far parte della rete. A quel punto, come stabilito da Symphony, questo deve assegnarsi un id e deve contattare il peer che lo amministra attualmente, attraverso un lookup, che deve partire da un peer noto a priori. Per questo motivo, ogni nodo dinamico, utilizza come peer noto un nodo casuale tra quelli statici. Tale scelta garantisce una uniformità nel carico in questa prima fase, poiché i punti di ingresso dei lookup sono uniformemente distribuiti all'interno dell'anello (ciò è vero poiché gli id dei peer statici è a sua volta uniformemente distribuito).

Un generico peer dinamico, dopo essere entrato all'interno della rete di overlay, rimane al suo interno fintantoché il churner non gli chiede di uscire. A quel punto il peer non deve far altro che eliminare tutti i collegamenti, ripristinando la topologia, e ritornare nella sua fase iniziale, attendendo nuovamente di poter rientrare.

Quando il peer dinamico si trova fuori la rete, può ricevere comunque dei messaggi (i.e. quelli che gli sono stati mandati poco prima la sua uscita dalla rete di overlay, e che quindi erano in transito al momento dell'uscita). Tali messaggi vengono scartati dal peer e rimandati al mittente, in modo da simulare fedelmente ciò che accadrebbe nella realtà (si pensi ai tempi di attesa nel caso in cui provassimo a contattare un peer che esiste, ma che non vuole accettare una connessione). In tal modo, il mittente può gestire tale messaggio nuovamente, inviandolo al peer più appropriato (si pensi ad esempio al lookup).

4.2.0.3 Churner. Il churner è un'entità esterna alla rete di overlay, che non partecipa quindi alla computazione. Ciò che viene fatto da tale entità è semplicemente far entrare (e far uscire) un peer per ogni unità di tempo (e.g. far entrare un peer ogni secondo, e farne uscire uno ogni due secondi). La scelta di creare quest'entità esterna è stata fatta per via del fatto che, lasciare la gestione di quando entrare (ed uscire) al peer stesso, non riesce a fornirci la stessa flessibilità. Infatti, gestire la frequenza di entrata e di uscita nei peer stessi, richiederebbe una complessa coordinazione tra i peer.

La scelta del peer da far entrare è casuale, tra quelli che sono fuori la rete di overlay. Allo stesso modo decide il peer da far uscire, tra quelli che sono all'interno della rete. Pertanto, in quest'ultimo caso, non viene mai detto ad un peer che sta ancora compiendo delle operazioni per entrare.

4.3 Livello di astrazione

La simulazione, come abbiamo precedentemente visto, è composta dai vari peer, collegati tra di loro mediante gli short link ed i long link. Una prima osservazione sul livello di astrazione è pertanto quella che riguarda la topologia della rete che collega tali peer. Nella realtà, ogni peer fa parte della rete reale di Internet, che è differente dalla rete di overlay logica creata da Symphony. Nella nostra simulazione, viene rappresentata solamente quest'ultima, dove ogni link (sia short che long) ha un delay di 100ms ed un bandwidth di 10Mbps, che stimiamo essere una buona approssimazione di un usuale canale di comunicazione logico, che colleghi due peer presenti sulla rete di Internet.

In una reale implementazione del protocollo, i peer che entrano ed escono dalla rete hanno bisogno di meccanismi di accesso in sezione critica distribuiti, per non creare delle incoerenze che potrebbero intaccare negativamente la struttura della rete di overlay. Tali meccanismi sono a volte complessi da implementare e non sono oggetto del nostro studio. Per risolvere tutti gli accessi a sezione critica è stato pertanto utilizzato il fatto che nella simulazione, in realtà, qualunque metodo (delle classi che implementano i vari moduli) non viene mai eseguito in concorrenza. Ciò significa che tutte le manipolazioni della topologia vengono fatte dal peer interessato, in un solo metodo, compiendo pertanto un'azione atomica. Per capire meglio, prendendo l'evento di joining del peer p , che vuole entrare con un id x , il manager, che amministra l'id x , sarà quello che modificherà gli short link per far entrare p all'interno della rete di overlay (ciò viene fatto istantaneamente, quando il manager riceve la richiesta di joining da parte di p).

Si assume che ogni peer all'interno abbia una conoscenza dei peer che compongono il proprio vicinato (due nodi per gli short link e meno di sei per i long link). Ciò è un'assunzione realistica comunemente utilizzata nei reali sistemi peer-to-peer. Ogni peer conosce pertanto l'id di ogni suo vicino, recuperato al momento della creazione del link che li ha connessi. Lo stesso non si può dire per gli id gestiti da un peer vicino, poiché in questo caso tale informazione cambia continuamente.

Ciò è necessario per chiarire due cose molto importanti. La prima è sapere quali sono gli id gestiti dal proprio peer: tale informazione viene ottenuta senza l'invio di nessun messaggio, considerando il proprio id e quello del peer precedente. La seconda è sapere a chi dobbiamo fare il forward di un messaggio di lookup che richiede di cercare il manager dell'id x . Anche in questo caso, senza mandare messaggi, possiamo semplicemente controllare gli id di ogni peer vicino, e fare il forward al peer con l'id più vicino ad x , rispettando le specifiche del Routing Protocol descritto da Symphony. In una reale implementazione queste due considerazioni continuerebbero a valere, rendendo questi aspetti fedelmente emulati.

4.3.1 Messaggi. Ogni peer può mandare diversi messaggi:

- *LookupMsg*: è il messaggio di lookup. Tale messaggio può essere utilizzato per tre motivi diversi: per cercare il manager di un id quando viene fatto un join, per cercare il manager di un id per poter creare un long link, oppure per un lookup generico che simula la ricerca di un file all'interno della rete.
- *LookupResponseMsg*: è la risposta ad un lookup, che trasporta al suo interno l'identificatore del manager. In questo modo il peer che ha fatto la richiesta può compiere un'azione opportuna.
- *NEstimationMsg*: è il messaggio che viene spedito per aggiornare la stima del numero di peer presenti all'interno della rete, come previsto dal protocollo di Symphony.

Ogni richiesta di lookup emessa da un determinato peer, viene identificata da un id crescente. Anche nel caso questo uscisse e rientrasse dalla rete, tale id è garantito essere distinto. Ogni peer mantiene poi una coda dei messaggi di lookup che sono stati inviati, associando ad ognuno di essi la procedura all'interno del peer che ha richiesto l'invio di tale messaggio. Ogni risposta ad un lookup (i.e. Looku-

pResponseMsg) mantiene al suo interno questi id, permettendo al peer, grazie alla coda, di identificare e continuare a svolgere la procedura che aveva richiesto il lookup (il messaggio viene ignorato se non si dovesse trovare nella coda una corrispondenza). Inoltre, considerando che la coda di messaggi viene svuotata al momento dell'uscita dalla rete di overlay, questo meccanismo permette anche di ignorare le risposte a lookup inviati in una precedente reincarnazione del peer, evitando il verificarsi di errori.

Per utilizzare i messaggi di lookup in modo da effettuare un join, il peer segna in maniera opportuna un campo all'interno del messaggio stesso, in modo da distinguerlo. Così facendo, il manager dell'id, invece di rispondere solamente a tale lookup, può manipolare la topologia della rete di overlay, in modo da far entrare il peer al suo interno. Le operazioni di join non possono infatti essere fatte dal peer stesso, poiché, durante il tempo trascorso per la ricezione della risposta del lookup, il manager può non gestire più l'id che gli era stato richiesto, generando conflitti che perturberebbero la topologia della rete di overlay. Il join, compiendo un'azione atomica per manipolare la topologia, utilizza pertanto un numero di messaggi che nella realtà sarebbe più alto.

L'operazione dei leave viene fatta dal peer in maniera atomica, anche qui sfruttando il concetto discusso precedentemente, in modo da evitare qualunque errore di concorrenza. Anche in questo caso, come il join, il numero di messaggi inviati è minore a quello che sarebbe in un protocollo reale.

5. RISULTATI

6. CONCLUSIONI