Una comparazione tra algoritmi di GOSSIP su differenti simulatori

Relazione del progetto di Simulazione di Sistemi

Daniele Baschieri, Michele Corazza, Stefano Gombi

23 settembre 2014

Sommario

In questa realazione vengono presentati i risultati del confronto tra simulatori, Lunes e una nostra implementazione basata su OmNET++. Le simulazioni, ispirate da "A fair comparison of gossip algorithms over large-scale random topologies"[1], riguardano tre diversi algoritmi di gossip (Probabilistic Broadcast, Probabilistc Edge, Fixed Fanout) simulati su tre diversi tipi di grafi (Erdős–Rényi, Random Geometric Graph, scale free). Tali simulazioni hanno lo scopo di verificare alcuni risultati che emergono dal paper originale, introducendo però variabilità nell'utilizzo di cache sui nodi, in modo da analizzare il comportamento del sistema anche in questo caso.
Viene inoltre effettuata una comparazione fra i due simulatori, sia per quanto riguarda alcune metriche che descrivono la diffusione dei messaggi, sia considerando il numero di eventi generati e il tempo di esecuzione.

1 Introduzione

L'utilizzo di algoritmi di gossip è legato alla necessità di diffondere un messaggio in reti di vario tipo minimizzando l'utilizzo dei canali di comunicazione. Mediante broadcast puro, infatti, si ottiene una propagazione certa del messaggio a tutti i nodi del grafo, causando però una saturazione dei canali di comunicazione. Pertanto sono stati progettati gli algoritmi di gossip, i quali si ispirano allo scambio di informazioni fra gli esseri umani. La comunicazione di informazioni fra esseri umani viene emulata tramite uno scambio di messaggi, regolato da una probabilità di trasmissione prefissata in forme diverse a seconda dell'algoritmo scelto. Fra gli algoritmi di gossip più utilizzati vi sono quelli da noi selezionati: Probabilist Broadcast (PB), Probabilistic Edge (PE) e Fixed Fanout (FF). Tali algoritmi utilizzano meccanismi differenti per gestire lo scambio di messaggi e sono pertanto complessi da confrontare. Per tale ragione il paper[1] in esame propone l'utilizzo dell'Effectual Fanout, un valore che è possibile calcolare a partire dalle probabilità di ciascun algoritmo, e che quindi permette una comparazione fra di essi. Tra i risultati più interessanti vi è anche la linearità fra l'EFF e la Message Complexity, una misura del numero totale di messaggi spediti che sarà approfondita successivamente. Le performance degli algoritmi dipendono inoltre dal tipo di grafo scelto. Per tale ragione si sono effettuate analisi su grafi Erdős–Rényi, Scale Free Graph e Random Geometric Graph, adatti a rappresentare rispettivamente reti P2P, reti generiche di calcolatori (es WWW, Internet) e reti wireless.
Il lavoro svolto a partire da tale contesto ha coinvolto due simulatori diversi, LUNES e un simulatore da noi implementato a partire da OmNET++. Le simulazioni svolte hanno riguardato la verifica dei risultati sperimentali del paper sui tre tipi di grafo e con i tre diversi algoritmi. Si è poi proceduto a variare la dimensione delle cache sui nodi per studiarne l'impatto. Oltre a tali analisi qualitative, si sono effettuate delle comparazioni fra i due simulatori, al fine di evidenziare discrepanze nella generazione di eventi o nel tempo impiegato.

2 Scenario

2.1 Gli algoritmi di gossip

Nel contesto di reti P2P di varia natura l'utilizzo di broadcast per consentire la diffusione di un messaggio a tutti i nodi coinvolti è un'operazione costosa, che tende a saturare rapidamente i canali di comunicazione, specialmente per reti di grandi dimensioni. Per ovviare a questo inconveniente sono stati studiati protocolli ad-hoc, che permettono, imitando la diffusione di notizie fra gli esseri umani, di propagare informazioni fra i nodi riducendo il numero di messaggi richiesti. Tali algoritmi, detti di *gossip*, utilizzano un valore probabilistico per decidere sulla diffusione di un determinato messaggio da parte dei singoli nodi.
Le simulazioni qui presentate riguardano, in particolare, tre algoritmi di gossip fra i più diffusi: Probabilistic Broadcast (PB), Probabilistic Edge (PE) e Fixed Fanout (FF).
Procediamo brevemente alla loro descrizione, utilizzando la seguente notazione:

- Λ_i: l'insieme dei nodi adiacenti all'i-esimo nodo;
- V_i: la cardinalità di Λ_i;
- msg: il messaggio ricevuto dal nodo;
- $p_v, p_e, fanout$: sono i valori probabilistici relativi a ciascun algoritmo, rispettivamente a PB,PE,FF.

2.1.1 Probabilistic broadcast

Algorithm 1 Probabilistic Broadcast

```
function GossipPB(msg,p_v)
    if Random() ≤ p_v then
        for all s_j ∈ Λ_i do
            Send(msg, s_j)
        end for
    end if
end function
```

Ciascun nodo che ha ricevuto un messaggio lo diffonde in broadcast a tutti i nodi adiacenti con una probabilità prestabilita.

2.1.2 Probabilistic Edge

Algorithm 2 Probabilistic Edge

```
function GossipPE(msg,p_e)
    for all s_j ∈ Λ_i do
        if Random() ≤ p_e then
            Send(msg, s_j)
        end if
    end for
end function
```

Ciascun nodo che ha ricevuto un messaggio lo diffonde a ciascun nodo adiacente con una probabità prestabilita procedendo in modo iterativo sugli archi.

2.1.3 Fixed Fanout

Algorithm 3 Fixed Fanout

```
function GossipFF(msg,fanout)
    if fanout ≥ V_i then
        toSend ← Λ_i
    else
        toSend ← ∅
        for f = 1 to fanout do
            random select s_j ∈ Λ_i/toSend
            toSend ← toSend ∪ s_j
        end for
    end if
    for all s_j ∈ Λ_i do
        Send(msg, s_j)
    end for
end function
```

Ciascun nodo che ha ricevuto un messaggio lo diffonde a *fanout* nodi adiacenti selezionati in modo casuale.

2.2 Tipologie di Reti

Il comportamento degli algoritmi di gossip appena descritti è fortemente influenzato dalla topologia delle reti su cui vengono eseguiti. Tali topologie presentano strutture differenti fra loro e possono essere utilizzate per rappresentare diversi scenari. Il paper preso in esame considera tre tipi di reti: Bernoulli graph, random geometric graph, scale-free graph. Poiché le reti *large-scale* fra calcolatori hanno solitamente canali di comunicazione bidirezionali tutti i grafi considerati sono non orientati. Ne forniamo di seguito una breve descrizione, considerando due caratteristiche dei grafi:

- La edge dependency o interdipendenza degli archi, definita come la probabilità subordinata che se esistono due archi $s_i \sim s_k$ e $s_j \sim s_h$, esista un arco $s_i \sim s_j$. Formalmente è la probabilità $P(s_i \sim s_j \mid s_i \sim s_k, s_j \sim s_k)$. Si dice alta se è maggiore della probabilità semplice $P(s_j \sim s_h)$.
- La degree variance, ovvero la varianza del grado dei nodi.

2.2.1 Bernoulli graph

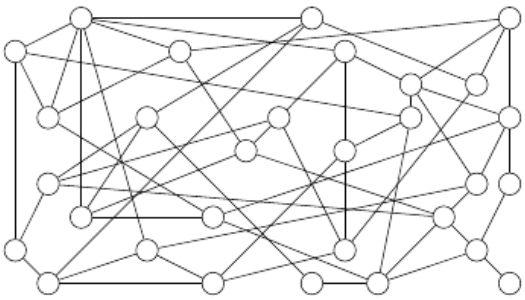

Figura 1: Un grafo di Bernoullli

Un grafo generato secondo il modello di Bernoulli (o di Erdős–Rényi) è costruito collegando ciascuna coppia di nodi con una probabilità pN. Tale costante positiva determina la probabilità con la quale viene creato un arco. Il modello di Bernoulli è particolarmente adatto per rappresentare una rete P2P di calcolatori, nella quale i collegamenti fra i nodi sono puramente casuali e ciascun nodo tende ad avere lo

stesso numero di archi incidenti.
Presenta bassa edge dependency e bassa degree variance.

2.2.2 Random Geometric Graph

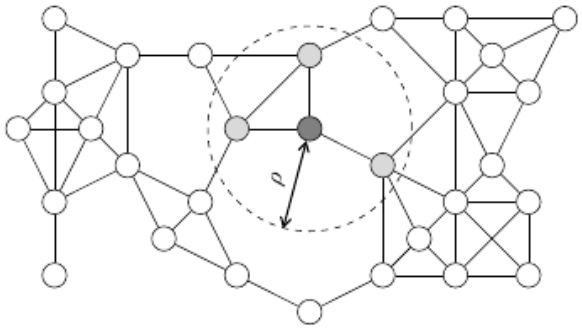

Figura 2: Un Random Geometric Graph

Un Random Geometric Graph viene generato utilizzando la prossimità geometrica in due dimensioni per determinare quali archi esistano fra i nodi. Dapprima vengono posizionati i nodi in uno spazio rettangolare, ricavando le due componenti delle coordinate cartesiane tramite un generatore di numeri casuali. Si procede poi confrontando le distanze fra i nodi. Se due nodi a e b hanno una distanza inferiore a una soglia fissata ρ viene generato l'arco (a,b) (si noti la presenza della distanza nella figura 2 che evidenzia i nodi prossimi a quello "centrale").
Appare evidente come un grafo generato tramite questo algoritmo possa essere utilizzato per descrivere reti wireless, caratterizzate dalla distanza fisica fra i nodi (trasmettitori/ricevitori) e da un valore ρ che rappresenta la soglia entro la quale sono possibili trasmissione e ricezione da parte dei nodi.
Presenta alta edge dependency (a causa della nozione di vicinanza geometrica usata per costruire i grafi) e bassa degree variance.

2.2.3 Scale-free Graph

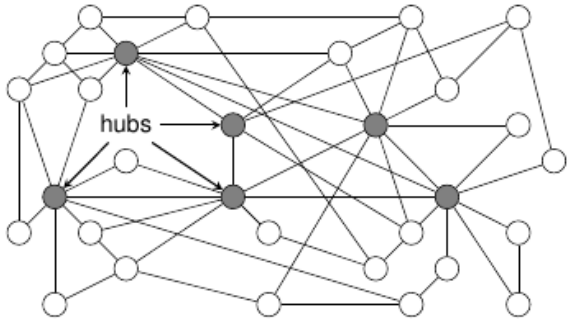

Figura 3: Un grafo scale-free

I grafi scale-free rappresentano reti con una degree distribution che si comporta asintoticamente come una power law per valori alti di k:

$$P(k) \sim k^{-\lambda} \tag{1}$$

Dove P(K) denota la probabilità che un nodo abbia k archi incidenti, mentre il valore di λ è tipicamente compreso fra 2 e 3.

Per generare un grafo scale-free si è utilizzato il modello di Barabási-Albert, che procede nel modo seguente:

1. Si genera una cricca di $m0$ nodi;
2. Si procede su ciascun nodo colleganodolo a $m \leq m0$ nodi con una probabità proporzionale al numero di archi ad esso incidenti.

Questa procedura crea dei nodi hub che hanno relativamente più archi incidenti rispetto alla media (si notino gli hubs evidenziati in grigio nella figura 3).
Un grafo scale-free è adatto a rappresentare molte reti differenti, quali la stessa internet, reti di citazioni, il World Wide Web, etc.
Presenta bassa edge dependency e alta degree variance (a causa della presenza di nodi hub).

2.3 Comparare algoritmi differenti: Effectual Fanout

In precedenza si sono descritti alcuni algoritmi di gossip con caratteristiche e parametri in input differenti. Se si desidera effettuare un'analisi prestazionale di tali algoritmi su topologie di rete diverse, appare evidente come i parametri in ingresso, che determinano la probabiità della disseminazione, abbiano caratteristiche troppo disomogenee per consentire una semplice comparazione. Pertanto è stata proposta[1] una metrica ad-hoc, chiamata Effectual Fanout (EFF), che permette di valutare la performance a prescindere dall'algoritmo utilizzato. La definizione di tale metrica dipende dall'algoritmo considerato. Per Probabilistic Broadcast, Probabilistic Edge e Fixed Fanout, EFF è definito rispettivamente come segue:

$$EFF_{PB} = p_v \bar{V} \tag{2}$$

$$EFF_{PE} = p_e \bar{V} \tag{3}$$

$$EFF_{FF} = \sum_{k=1}^{fanout-1} P(k) * k + \sum_{k=fanout}^{N-1} P(k) * fanout \tag{4}$$

Dove:

- $\bar{V}$ è il grado medio del grafo, ovvero il numero medio di archi incidenti a un nodo;
- p_v è la probabilità passata come input a Probabilistic Broadcast;
- p_e è la probabilità passata come input a Probabilistic Edge;
- $fanout$ è il valore intero passato come input a Fixed Fanout;
- $P(k)$ è la probabilità che un nodo abbia k archi incidenti.

Tale metrica permette dunque una comparazione fra gli algoritmi, a patto di conoscere il grado medio e la degree distribution[1] del grafo considerato. È inoltre possibile comparare i comportamenti degli algoritmi su grafi differenti, a patto che essi abbiano lo stesso grado medio. Per tale ragione il generatore da noi utilizzato produce grafi di 500 nodi con grado medio 8 e grafi di 1000 nodi con grado medio 14 per tutte le topologie in esame.
Un altro aspetto interessante dell'effectual fanout è il suo rapporto con la Message Complexity M, una metrica definita come segue:

$$M = \frac{\Omega}{N-1} \tag{5}$$

Dove:

- Ω è il numero totale di messaggi inviati/ricevuti (non vi è perdita di messaggi) dai nodi.
- N è il numero di nodi del grafo

Mediante un teorema[1] è possibile dimostrare che per valori sufficientemente elevati di N la Message Complexity cresce linearmente con l'Effectual Fanout.

[1] I P(k) definiti in precedenza per k compreso fra 1 e il grado massimo del grafo

2.4 Generazione di grafi e grado medio

Abbiamo visto come il grado medio sia fondamentale per comparare grafi di topologie differenti tramite EFF. Per tale ragione è necessario discutere alcune assunzioni riguardanti tale caratteristica dei grafi. Se consideriamo i nodi di un grafo $\{s_1, s_2, ..., s_n\}$ e chiamiamo Λ_i l'insieme dei nodi adiacenti ad s_i, $V_i =\mid \Lambda_i \mid$ e P(k) è la frazione di nodi con grado $V_i = k$. Ne segue che il grado medio del grafo $\bar{V}$ è dato da $P(K) = \sum_{k=0}^{N-1} P(k)k$.
Questo ci consente di procedere come segue sulle tre topologie per produrre grafi con grado medio prestabilito. Per quanto riguarda grafi di Bernoulli, supponiamo che la probabilità di collegamento di due nodi $pN > \frac{(1+\epsilon)ln(N)}{N}$ con ϵ costante positiva. Ciò conduce ad una componente gigante con N nodi aventi degree distribution secondo la legge di Poisson: $P(k) = exp(-\bar{V})\frac{\bar{V}^k}{k!}$ dove il grado medio è dato da: $\bar{V} = pN * N$.
Procedendo analogamente su un grafo di tipo Random Geometric, dove vengono generati punti su un rettangolo bidimensionale di dimensioni (a,b), possiamo imporre che il raggio $\rho > \sqrt{\frac{(1+\epsilon)ln(n)(ab)}{N\pi}}$ con ϵ costante positiva ottenendo un grafo connesso con una degree distribution secondo la legge di Poisson $P(k) = exp(-\bar{V})\frac{\bar{V}^k}{k!}$, con $\bar{V} = \frac{N\pi\rho}{ab}$.
Per quanto riguarda i grafi Scale-free, si aggiungono m connessioni ad un nodo a partire da una cricca di cardinalità m_0. Questo processo ci assicura che il grafo abbia una degree distribution che segue una power law, approssimata da $P(K) = \frac{2m(m+1)}{k(k+1)(k+2)}$ dove k = m, m+1, ... , N-1 e $\bar{V} = 2m$, che non dipende dalle dimensioni del grafo.
Tramite tali relazioni è dunque possibile generare grafi con grado medio prestabilito semplicemente risolvendo le equazioni fornite.

3 Obiettivi

3.1 Valutare le prestazioni degli algoritmi tramite EFF

Sebbene la proposta di EFF risulti interessante, nell'articolo vengono effettuate simulazioni che a nostro avviso non rappresentano in modo esaustivo reali reti di calcolatori. Il meccanismo di diffusione proposto non viene infatti descritto in maniera esaustiva: due aspetti fondamentali quali l'utilizzo eventuale di meccanismi di caching sui nodi e di Time To Live (TTL) dei messaggi non vengono menzionati. Un aspetto correlato alla presenza di cache riguarda il meccanismo di generazione dei messaggi: se essa fosse infinita la tempistica con la quale viene generato ciascun messaggio non influenzerebbe i risultati finali (ciascuna diffusione sarebbe indipendente dalle altre), con cache limitata e una generazione di messaggi eccessivamente frequente si avrebbe invece una rapida saturazione della memoria sui nodi. A nostro avviso lo scenario più realistico è quello di un meccanismo di diffusione nel quale ogni nodo ha una memoria limitata e vi sono vincoli temporali per la validità dei messaggi. Al fine di studiare tali aspetti si sono effettuate simulazioni su LUNES e un simulatore basato su Omnet++ da noi sviluppato. Tali simulatori hanno la caratteristica di considerare una cache limitata, che contiene gli ultimi messaggi ricevuti. In questo modo solo se un messaggio già ricevuto è ancora in cache verrà interrotta la sua diffusione, simulando di fatto una rete che è influenzata dal numero di messaggi in transito. Viene inoltre considerato un TTL per limitare la vita dei messaggi all'interno della rete, pari al grado massimo di ciascun grafo.

3.2 Confrontare i simulatori

Il secondo obiettivo consiste nella valutazione prettamente prestazionale dei due simulatori. LUNES e il nostro simulatore Omnet presentano lo stesso meccanismo di generazione di messaggi, basato su una distribuzione esponenziale di media 10 che viene utilizzata per determinare i timestep fra due generazioni di messsaggi successive su ciascun nodo. I timestep di OmNet e LUNES vengono limitati da un parametro apposito, che ne determina il valore massimo. Vengono naturalmente considerate le stesse reti sui due simulatori. La valutazione prestazionale verrà effettuata mediante due metriche: da una parte il numero di eventi totali generati da ciascun simulatore, dall'altra il tempo effettivo di esecuzione.

3.3 Simulatori ed eventi

Confrontare il numero di eventi di LUNES ed OmNet++ è un'operazione complessa, in quanto i risultati ottenuti dipendono fortemente dalla semantica degli eventi propria a ciascun simulatore. Nel nostro caso, se in Omnet++ viene riportato il numero di eventi totali in output, LUNES non produce tale valore. Al fine di effettuare una comparazione ragionevole, si è scelto di utilizzare la stessa semantica per gli eventi in entrambi i simulatori. Si è dunque deciso di contare il numero di messaggi ricevuti e generati durante una simulazione della stessa durata. Naturalmente, poiché i simulatori generano i messaggi allo stesso modo, ci si attende una sostanziale uguaglianza nel numero di eventi generati. Tale uguaglianza ci consente di effettuare confronti prettamente temporali fra i simulatori senza temere che vi siano differenze sostanziali nel loro comportamento.

4 Strumenti

4.1 LUNES: Un simulatore ad agenti di reti P2P

LUNES (Large Unstructured Network Simulator)[2] è un simulatore basato su agenti che consente di eseguire algoritmi di gossip su reti composte da un alto numero di nodi. LUNES ha una struttura modulare, che consta di tre software distinti, i quali consentono di rendere indipendenti le tre fasi di creazione del grafo, siulazione ed analisi dei dati. In particolare, i grafi possono essere generati tramite l'utilizzo della libreria igraph, a patto di esportarli nel formato graphviz. I servizi di simulazione sono forniti dal middleware ARTIS e dal framework GAIA. In particolare, ARTIS fornisce le primitive che riguardano la sincronizzazione e la comunicazione fra nodi, mentre GAIA fornisce funzionalità avanzate di load-balancing fra i Logical Process (LP). Tramite tali middleware, è possibile effettuare simulazioni distribuite o in parallelo su macchine multiprocessore.
Poiché paralelizzare le simulazioni su protocolli di gossip può condurre a speedup minori di uno (simulazioni parallele più lente di quelle sequenziali) GAIA implementa metodi adattivi per modificare la partizione dei nodi fra i LP, riducendo le comunicazioni fra di essi.
La simulazione procede per passi discreti, nei quali ciascun nodo propaga i messaggi che ha ricevuto o ne genera di nuovi. Viene naturalmente posto un limite massimo al numero di passi da eseguire. La generazione dei messaggi in LUNES avviene in modo casuale: ciascun nodo del grafo genera un messaggio ad un timestep selezionato tramite un generatore di numeri random, che utilizza una distribuzione esponenziale.

4.2 OMNeT++

OMNeT++ è un framework ed una libreria per la simulazione, principalmente volta alla creazione di simulatori di reti. Il comportamento delle componenti di rete (es: Nodi,Archi) viene definito in C++, tramite classi di libreria prefenite o implementate dallo sviluppatore. La struttura della rete viene invece descritta da file NED, contenenti ad esempio i collegamenti fra nodi di un grafo. Tramite l'IDE fornito con OMNeT++ è possibile importare grafi nel formato graphml e convertirli in file NED. Anche la definizione della struttura dei messaggi avviene tramite un file ad-hoc, che viene poi trasformato in una classe C++ da un apposito tool.
Il simulatore utilizza un kernel real-time per fornire una nozione di tempo di simulazione e primitive di sincronizzazione temporale, da noi utilizzate per emulare il comportamento di LUNES per quanto riguarda la generazione di messaggi e il ritardo di trasmissione.

5 Implementazione

5.1 Modifiche a LUNES

Per poter simulare gli scenari proposti dal paper è stato necessario apportare alucne modifiche a LUNES, sia per quanto riguarda la fase di simulazione sia per la fase di analisi dei dati.
Innanzitutto si è implementato Fixed Fanout, per cui erano già state definite le opportune variabili d'ambiente ma mancava l'algoritmo vero e proprio.
Si è poi affrontato il problema di valutare gli algoritmi di gossip a partire dalla metrica EFF proposta,

modificando gli script incaricati di eseguire le simulazioni di LUNES. A tal fine, per quanto riguarda gossipPB e gossipPE si è scelto di procedere iterativamente incrementando il valore di EFF fino a giungere il grado del grafo. L'approccio intrapreso con FF è stato invece differente. Poichè non è possibile risalire usando la formula 4 dal valore di EFF al fanout, che è necessario all'algoritmo, abbiamo preferito procedere iterativamente su fanout, calcolando poi i valori medi di EFF. Si sono inoltre implementate delle piccole utility per calcolare grado medio, degree probability, grado massimo e diametro dei grafi.
Per generare grafi della tipologie specificate dal paper, è stato scritto un generatore di grafi basato sulla libreria igraph, in grado di produrre random geometric graph, bernoulli graph e scale-free graph con un grado medio costante. Tale generatore è stato poi esteso per fornire output anche in formato graphML, in modo da consentirne l'interoperabilità con OMNeT++.
Tramite le analisi dei dati disponibili in LUNES non era possibile calcolare alcuni valori interessanti, quali ad esempio la latenza dei messaggi (intesa come latenza dei messaggi consegnati a tutti i nodi).Un ulteriore problema riscontrato concerne il consumo di risorse dell'analisi dei dati implementata in LUNES. Tale analisi, infatti, ha un costo in termini di spazio in memoria molto elevato. A causa di ciò la memoria RAM dei PC a nostra disposizione veniva esaurita per tracce di dimensioni elevate, causando l'utilizzo costante di swap e il conseguente peggioramento delle prestazioni. Si è quindi deciso di fornire un'implementazione alternativa, basata su hash table e memoria dinamica. Per memorizzare i messaggi ricevuti si è utilizzata una prima hash table, mentre per tenere traccia delle coppie nodo-messaggio si è utilizzata un'altra hash. Al fine di memorizzare due interi (l'id del nodo e del messaggio) in un un unico valore da cui si potessero recuperare i dati di partenza, si è scelto di utilizzare una mappatura di due interi a 32 bit in un intero a 64 bit. Poichè la funzione di Cantor non garantisce che le dimensioni dell'output siano doppie rispetto alle dimensioni dei singoli interi in input, si è optato per la funzione di Szudzik[3], che rispetta invece tale vincolo. Per migliorare ulteriormente l'efficenza delle analisi si sono snelliti i file di traccia, riducendone le dimensioni. Il consumo di memoria dell'analizzatore di dati si è rivelato molto inferiore a quello preesisente.

5.2 Simulatore OMNeT++

L'implementazione del simulatore OMNeT++ è stata effettuata in modo quanto più simile a LUNES possibile, in modo da consentire una comparazione ragionevole fra i due. La generazione dei messaggi, analogamente a LUNES, avviene tramite un generatore di numeri casuali che segue una distribuzione esponenziale per determinare il tempo fra la generazione di un messaggio da parte di un determinato nodo e quella successiva. La distinzione fondamentale è invece data dal modo differente fra i simulatori di tenere traccia dello scorrere del tempo. Se LUNES ha un clock di simulazione discreto, in cui ad ogni passo vengono effettuate in contemporanea tutte le operazioni schedulate per quell'istante, OMNeT++ utilizza una misura temporale floating point ed uno scheduler real-time. Dai test effettuati tali approcci non conducono però a differenze significative, né nel numero di messaggi generati né nei risultati ottenuti.
Per implementare una cache si sono utilizzate doubly-linked list della libreria standard di C++, con un algoritmo Least Recently Used (LRU) per scegliere l'entry da rimuovere per in caso di inserimenti con cache piena.
Per consentire una comparazione che tenesse conto solo del tempo di simulazione e non di quello di analisi dei dati, si è scelto di produrre con OmNET++ tracce nel formato di LUNES, che vengono poi esaminate dall'analizzatore da noi implementato.

6 Analisi dei risultati

Si sono effettuate svariate simulazioni sulle tre topologie descritte e con i tre algoritmi di gossip in esame, al fine di evidenziare eventuali discrepanze di comportamento fra l'articolo e i simulatori in esame. Si sono inoltre valutate le prestazioni dei due simulatori, considerando il numero di eventi generati e le principali metriche riguardanti la diffusione dei messaggi. Tutti i test effettuati sono stati ripetuti su 10 grafi di ciascuna topologia, calcolando poi la media dei valori in esame.

6.1 Metriche

Al fine di analizzare l'andamento delle simulazioni, si sono utilizzate delle metriche che catturano diversi aspetti tipici della diffusione di messaggi in una rete:
Message Complexity: la message complexity, già discussa nella sezione 2.3;
Coverage: il rapporto fra il numero di messaggi ricevuti da ciascun nodo, escludendo ricezioni multiple dello stesso messaggio, e il prodotto del numero di messaggi generati per il numero di nodi;
Reliability: la frazione di messaggi ricevuti da tutti i nodi.
Tempo: il tempo di esecuzione della simulazione (escludendo l'analisi dei dati) è stato utilizzato per comparare le prestazioni dei simulatori.

6.2 LUNES: cache 256

Abbiamo dapprima considerato il comportamento dei tre algoritmi su grafi di 500 nodi, con grado medio 8 e cache di 256 messaggi. Si è utilizzato il simulatore LUNES, con una durata di simulazione di 1000 cicli. Per ciascuna esecuzione si sono misurate la Message Complexity, la Reliability e il Coverage in rapporto all'effectual fanout proposto dall'articolo. Consideriamo dapprima il comportamento dei tre algoritmi di gossip (Probabilistic Broadcast, Probabilistic Edge e Fixed Fanout) su grafi di tipo Erdos-Renyi(ER):

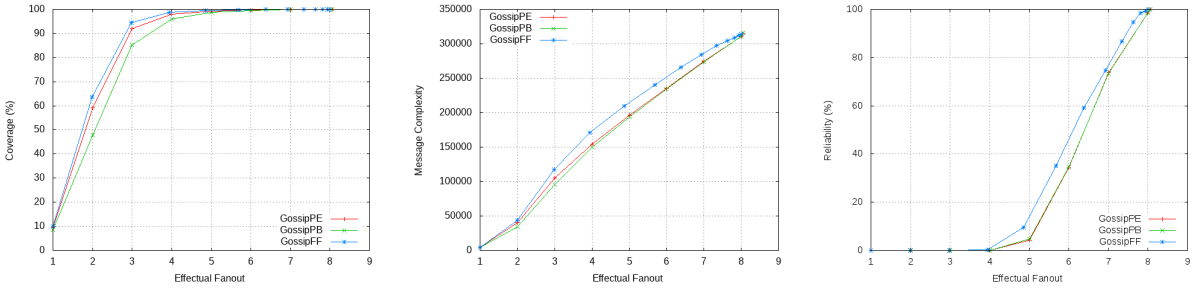

Figura 4: Coverage, Message Complexity e Reliability su grafi ER

Appare subito evidente osservando il grafico al centro della figura 4 come il rapporto fra l'effectual fanout e la Message Complexity sia sostanzialmente lineare, come previsto dall'articolo preso in esame. Per quanto riguarda il Coverage (sulla sinistra) si osserva una crescita molto rapida in rapporto all'effectual fanout, che evidenzia come anche valori di EFF relativamente bassi provochino una larga diffusione di messaggi. La Reliability (sulla destra) è tuttavia molto più ridotta rispetto al Coverage, in quanto solo probabilità molto alte di propagazione (e quindi EFF alto) garantiscono una diffusione dei messaggi a tutti i nodi. I grafi evidenziano inoltre una maggiore efficienza di Fixed Fanout, sia per quanto riguarda il Coverage che per la Reliability.
Passiamo ora a discutere i risultati si grafi scale-free:

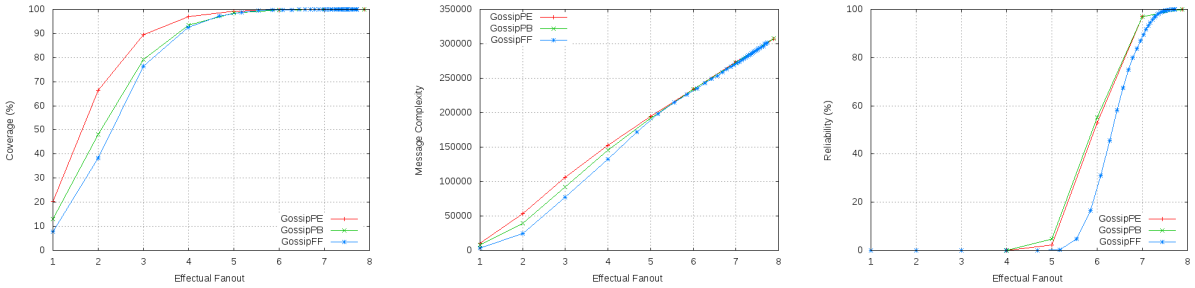

Figura 5: Coverage, Message Complexity e Reliability su grafi scale-free

I risultati appaiono simili a quelli di ER, con una relazione lineare fra Message Complexity ed EFF. Si ha un andamento simile di Probabilistic Edge e Probabilistic Broadcast per quanto riguarda la Reliability, con un leggero vantaggio per quest ultimo, mentre si ha un peggior rendimento di Fixed Fanout. I valori

di Coverage di Probabilistic Edge e Probabilistic Broadcast sono invece più distanziati, con un miglior risultato di PE . Entrambi si comportano comunque meglio di FF.
I risultati più interessanti emergono però dai grafi con topoplogia Random Geometric(rgeom), che hanno comportamenti molto discordanti con i risultati del paper:

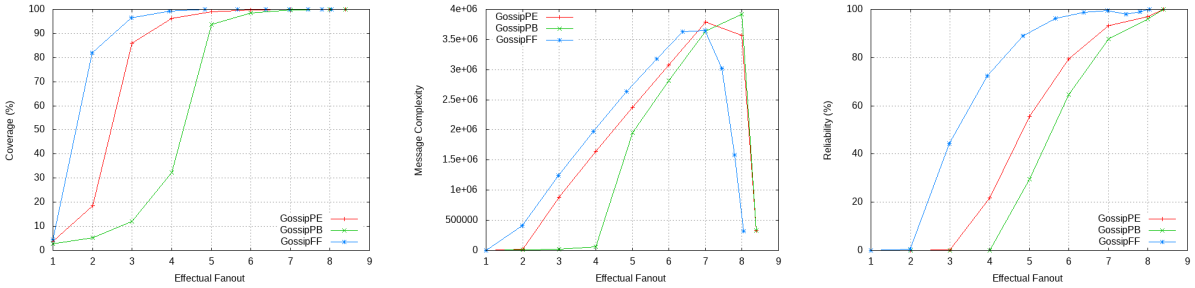

Figura 6: Coverage, Message Complexity e Reliability su grafi rgeom

Osservando il grafico al centro, appare evidente come la Message Complexity (e quindi il numero di messaggi nella rete) abbia un crollo verticale quando la probabilità di diffusione si avvicina al 100%. Le origini di tale anomalia sono da ricercarsi nell'alta inerdipendenza degli archi, in quanto in un grafo rgeom si osserva una sorta di vicinanza geometrica dei nodi, che sono connessi in base alla loro prossimità su un piano bidimensionale. Pertanto quando gli algoritmi diventano di puro flooding, le cache diventano molto più efficienti nel prevenire scambi rindondanti di messaggi, poiché nodi prossimi ricevono lo stesso messaggio più volte in tempi (di simulazione) molto brevi. Per quanto riguarda la comparazione fra gli algoritmi, si evidenzia un'efficenza maggiore di FF, legata alla bassa degree variance del grafo, in modo analogo a quanto osservato su grafi di tipo erdos-renyi. Osservando il grafico relativo alla reliability, si osserva addirittura un calo per valori di EFF alti. Tale flessione è legata ad uno solo dei grafi presi in esame, che in corrispondenza del crollo del numero di messaggi presenta un calo sostanziale (del 10% circa) anche nella reliability, probabilmente anch'esso legato ad un miglior funzionamento delle cache. Per studiare ulteriormente il crollo del numero di messaggi,si sono effettuati ulteriori test con LUNES su grafi rgeom variando la dimensione della cache, discusse in seguito.
Dai primi dati, le assunzioni del paper appaiono verificate, fatta eccezione per il rapporto fra la message complexity e l'EFF nei grafi di tipo random geometric.

6.3 Probabilist Broadcast e Fixed Fanout su random geometric graph con cache variabile

Al fine di comprendere meglio il fenomeno di calo della message complexity nel caso di grafi random geometric, si è pensato di effettuare test con cache di 128, 256 e 512 messaggi, in modo da evidenziare l'effetto della cache sulla message complexity, su coverage e reliability. Si è dapprima effettuato tale test con l'algoritmo Probabilistic Broacast:

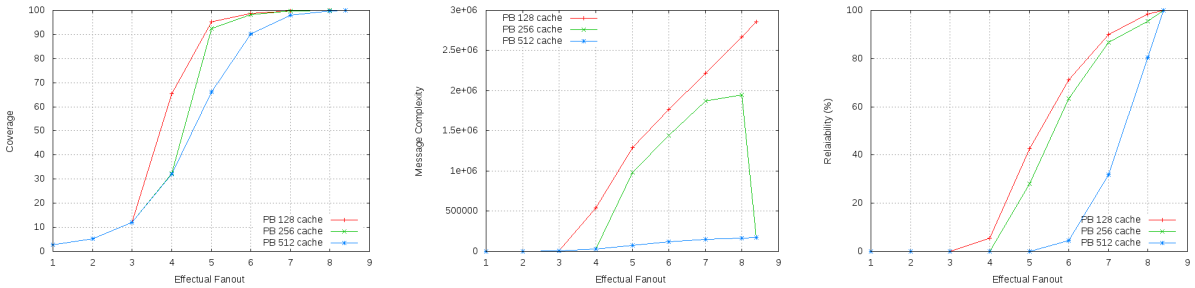

Figura 7: Coverage, Message Complexity e Reliability su grafi random geometric con cache variabile, algoritmo Probabilistic Broadcast

Dai grafici che riguardano il coverage (a sinistra della figura 7) e la reliability (a destra) emerge una relazione fra la diffusione dei messaggi e la cache. In particolare, cache di minori dimensioni corrispondono a una maggiore diffusione dei messaggi nella rete, come del resto ci si poteva attendere. Nel grafico centrale, che riguarda la Message Complexity, appare evidente una correlazione fra le dimensioni della cache e il calo del numero di messaggi che attraversano la rete. Per cache di 128 messaggi, infatti, tale calo non avviene, neppure per valori di EFF molto elevati. Una cache di 256 elementi, invece, presenta il calo di messaggi osservato in precedenza per valori di EFF alti. Con cache di 512, infine, il grafico non presenta il calo di messsaggi ma valori di message complexity molto inferiori. Osserviamo ora il comportamento di Fixed Fanout con cache variabile:

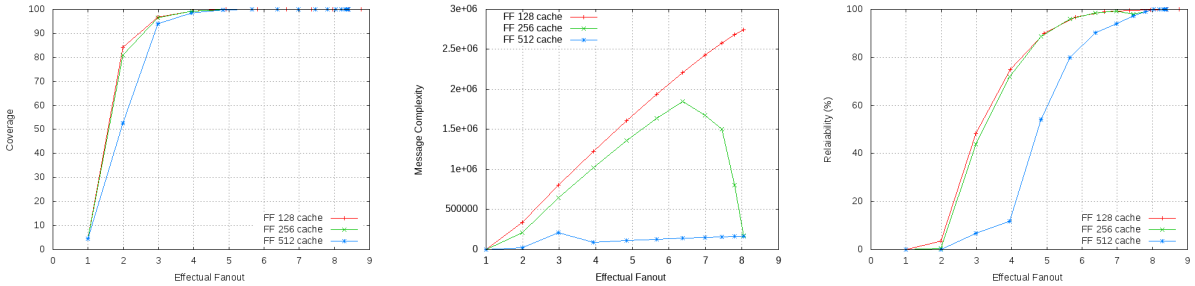

Figura 8: Coverage, Message Complexity e Reliability su grafi random geometric con cache variabile, algoritmo Fixed Fanout

Il comportamento di Fixed Fanout al variare delle cache appare molto simile a Probabilistic Broadcast. Cache inferiori infatti causano migliori reliability e coverage, mentre si osserva lo stesso calo nel numero di messaggi per cache di 256 elementi. Una particolarità è invece il comportamento per cache di 512, che presenta un calo nel numero di messaggi in corrispondenza di EFF=4. Per meglio evidenziare la discrepanza nell'andamento della Message Complexity sui due algoritmi con cache di 512 elementi, li compariamo mediante un grafico:

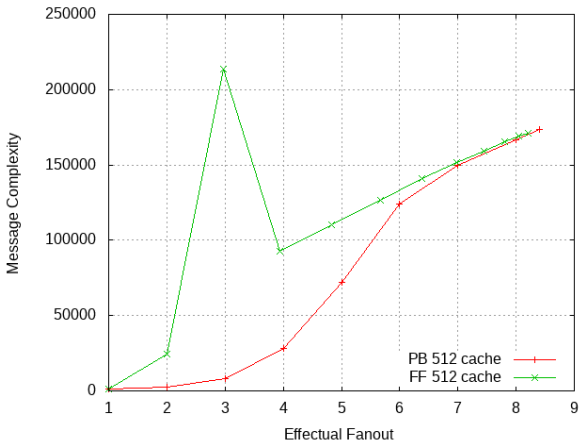

Figura 9: Message Complexity su grafi random geometric con cache di 512 messaggi, FF e PB

La figura 9 evidenzia come, se nel caso di PB non vi sia un calo nel numero totale di messaggi, nel caso di FF tale calo sia evidente fra EFF di valori 3 e 4. Si noti inoltre come la crescita del numero di messaggi per PB risulti molto più lenta rispetto ad FF. Tale comportamento dipende probabilmente dalla natura dei due algoritmi: se nel caso di PB i nodi spediscono burst di messaggi in broadcast, con l'effetto di una migliore resa delle cache, nel caso di FF si hanno spedizioni di messaggi in numero costante da ciascun nodo, che producono un aumento verticale nel numero di messaggi prima che le cache entrino a regime.
Si è dunque osservato come l'effetto della cache sulla message complexity sia determinante, in quanto essa produce effetti in contrasto con le previsioni del paper. La scelta dell'algoritmo è anch'essa influente,

in particolare per quanto riguarda cache di 512 elementi, per le quali la Message Complexity presenta andamenti differenti fra PB ed FF.

6.4 Random geometric di 1000 nodi

In seguito ai risultati ottenuti in precedenza, si è ipotizzato che il comportamento anomalo della Message Complexity con cache di dimensione relativamente alta dipendesse da un numero di nodi non sufficenti per evidenziare i risultati del paper. Si è pertanto deciso di effettuare una simulazione con un grafo random geometric con 1000 nodi e una cache di 512 messaggi usalndo l'algoritmo PB.

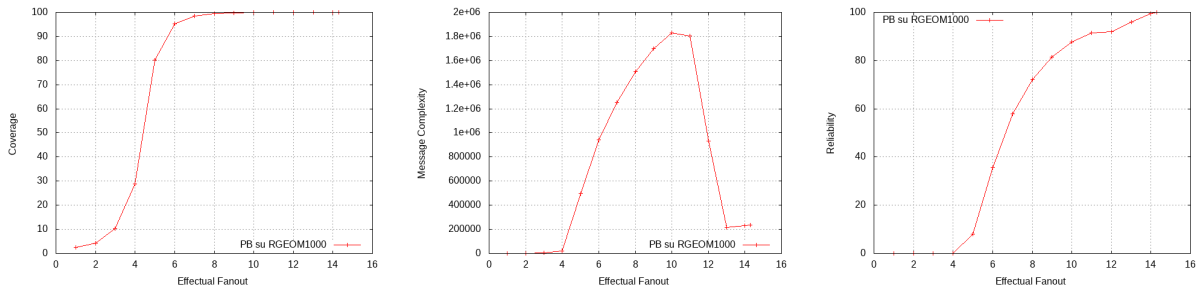

Figura 10: Coverage, Message Complexity e Reliability su grafi random geometric di 1000 nodi, algoritmo PB

I grafi di figura 10 mostrano un comportamento assimilabile a quello di figura 8 per cache di 256 elementi. Per quanto riguarda la Message Complexity, per valori di EFF elevati si osserva lo stesso calo nel numero di messaggi totali nella rete. Come in precedenza, inoltre, vi è una flessione nella crescita della reliability in corrispondenza del crollo del numero di messaggi. Se nel caso di cache di 256 e grafi di 500 nodi era ravvisabile addirittura una flessione nella reliability, in questo caso si osserva solo una crescita inferiore.
Da tali osservazioni emerge una relazione esistente fra il numero di nodi e la dimensione della cache. All'aumentare del numero di nodi, una stessa cache ha infatti un comportamento simile ad una cache di dimensione maggiore su un grafo di un numero minore di nodi.

6.5 Comparazione fra OmNET e LUNES

Al fine di comparare il simulatore LUNES con la nostra implementazione proof of concept basata su OmNET si sono misurate le usuali metriche di Coverage e Message Complexity. Per valutare il numero di eventi generati dai due simulatori si è scelto di considerare come evento ciascun messaggio inviato/ricevuto. Pertanto non si è graficato il numero di eventi in quanto tale, pochè esso è dato dal prodotto della message complexity per una costante (il numero di nodi).

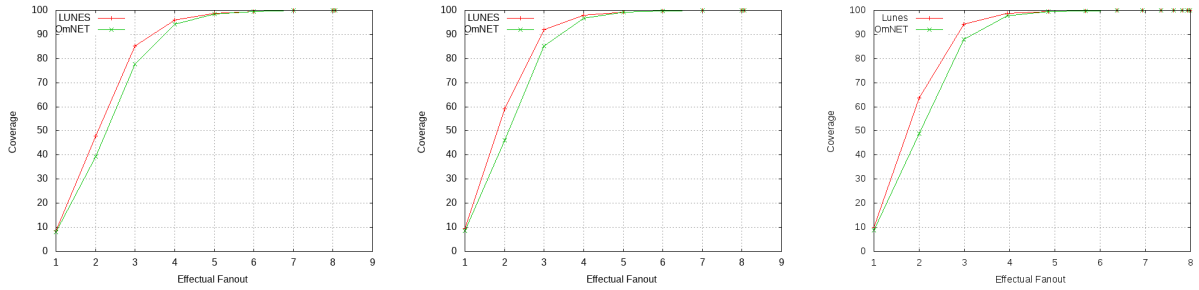

Figura 11: Coverage su grafi ER di 500 nodi, omnet e lunes con PB, PE ed FF

Il comportamento dei due simulatori appare molto simile, con valori di coverage leggermente più elevati nel caso di LUNES.

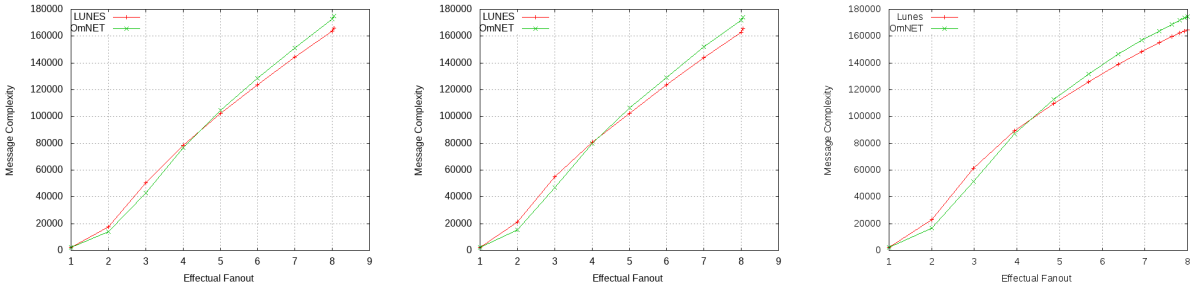

Figura 12: Message Complexity su grafi ER di 500 nodi, omnet e lunes con PB, PE ed FF

Anche nel caso della Message complexity l'andamento dei due simulatori appare quasi identico. Vi è però una piccola discrepanza, con un numero maggiore di messaggi su LUNES per EFF circa inferiore a 5 ed una situazione inversa per EFF maggiore.

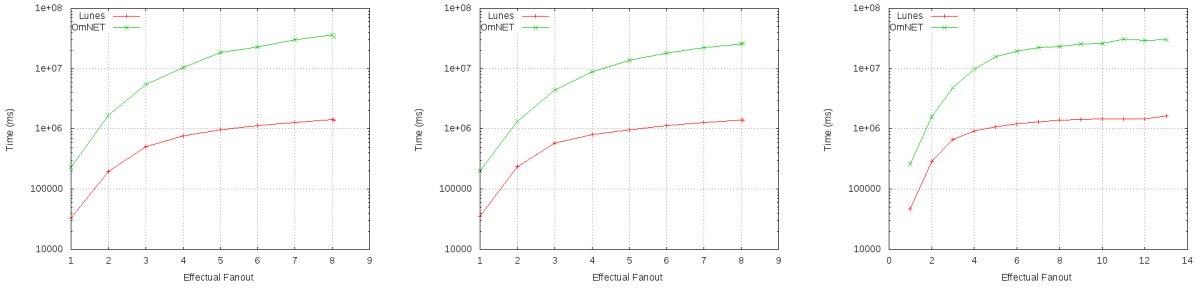

Figura 13: Tempo di esecuzione su grafi ER di 500 nodi, omnet e lunes con PB, PE ed FF

Il confronto temporale fra i due simulatori mostra un andamento simile al variare dell'EFF, ma è impietoso da un punto di vista generale: la scarsa ottimizzazione del simulatore OmNET lo conduce a tempi di esecuzione che sono svariate volte maggiori di quelli di LUNES.
I risultati ottenuti mostrano dunque comportamenti molto simili con tutti gli algoritmi. Vi è invece una differenza sostanziale nelle prestazioni, probabilmente a causa della mancata ottimizzazione del codice OmNET.

7 Sviluppi futuri

Uno degli elementi più significativi nelle nostre simulazioni è il comportamento degli algoritmi al variare della cache. Abbiamo potuto osservare come nelle figure 7 e 8 a seconda della dimensione della cache si ottengono comportamenti significativamente differenti, in particolare per quanto riguarda la message complexity. Per meglio comprendere i risultati sperimentali e il comportamento degli algoritmi sarebbe interessante simulare ulteriori scenari con valori diversi di cache, arrivando fino agli estremi di un sistema senza cache e un sistema con cache infinita. Il caso di un sistema con cache molto grande è più realistico rispetto ai nostri test in quanto nei dispositivi attuali il basso costo di questa componente la rende disponibile in larghe quantità. Questo caso sarebbe interessante per studiare la relazione fra il numero di nodi e la dimensione della cache, verificando la nostra ipotesi che al crescere di quest'ultima si mantenga una relazione lineare fra message complexity e effectual fanout. Dai nostri risultati emerge infatti un rapporto lineare fra EFF e message complexity nei casi con cache di dimensione relativamente molto bassa e molto alta, mentre si osserva un crollo del numero di messaggi nei casi intermedi. La dimensione della cache per cui avviene un crollo sembra però dipendere dal numero di nodi nel grafo; ulteriori test consentirebbero di comprendere meglio tale fenomeno. Sarebbe interessante anche verificare se questo comportamento si manifesti anche in grafi con bassa interdipendenza fra gli archi (diversi da RGEOM). Un altro aspetto rilevante che non è stato analizzato nel dettaglio riguarda il meccanismo di generazione dei messaggi, che nel nostro caso avviene con una distribuzione esponenziale di media 10. Aumentando o diminuendo tale valore è possibile rispettivamente migliorare o peggiorare il comportamento delle cache,

più o meno stressate dalla generazione di messaggi.
Un altro sviluppo possibile è il miglioramento del nostro simulatore basato su OmNET dal punto di vista prestazionale: in primo luogo sarebbe interessante reimplementare il meccanismo di caching con delle hash table. Inoltre sarebbe possibile migliorare la gestione dei canali di comunicazione fra i nodi, utilizzando anche in questo caso strutture dati più efficienti. In generale, la nostra comprensione ancora superficiale del simulatore OmNET ci fa sospettare che sia possibile ottenere prestazioni migliori operando delle ottimizzazioni al codice.
Una porzione non indifferente del tempo richiesto per le simulazioni è legata all'analisi di tracce di grandi dimensioni generate dai simulatori. Sebbene la nostra implementazione dell'analizzatore di tracce sia più efficente di quella già presente in LUNES, sarebbero possibili ulteriori ottimizzazioni. In particolare, il software utilizza due hash table, una indicizzata dall'id del messaggio e una legata alle coppie univoche nodo-messaggio. Si potrebbero migliorare le prestazioni annidando la seconda cache nella prima e usando come chiave solo l'id del nodo. Creando una doppia hash table, si ridurrebbero sensibilmente i tempi di inserzione e ricerca per le coppie nodo-messaggio, in quanto tali operazioni avverrebbero su hash table di dimensioni ridotte.

8 Conclusioni

Grazie alle nostre simulazioni abbiamo potuto osservare diversi comportamenti interessanti. Siamo riusciti a riprodurre degli scenari analoghi a quelli proposti dal paper, ottenendo i risultati previsti per quanto riguarda i grafi ER e SFREE. Sono tuttavia emerse delle lacune nell'esposizione da parte del paper riguardo alcuni meccanismi della simulazione, in particolare la presenza e dimensione di cache e il valore di Time To Live dei messaggi. Anche se possono sembrare aspetti secondari in realtà sono di primaria importanza per queste simulazioni in quanto in uno scenario in cui sono entrambi assenti la diffusione dei messaggi non avrà mai fine, portando il simulatore a divergere.
Il variare di tali grandezze porta inoltre a comportamenti qualitatimente differenti della simulazione. Come spiegato precedentemente, l'utilizzo di cache all'interno dei nodi sembra contraddire i risultati esposti dagli autori nel caso di grafi RGEOM, portando in certi casi a una caduta del numero di messaggi per valori di EFF relativamente alti. Questo comportamento si verifica per determinati valori del rapporto fra il numero di nodi e la dimensione della cache. Le simulazioni da noi compiute però non ci permettono ancora di stabilire con certezza quale sia il rapporto fra queste due grandezze per il quale si produce un crollo, anche se si è osservato che raddoppiando entrambe le quantità il fenomeno continua a manifestarsi.
Se tali risultati fossero confermati ciò implicherebbe che su alcuni grafi di tipo RGEOM con cache di dimensione intermedia (nel nostro caso con 500 nodi e cache di 256 messaggi) non vi sarebbe alcun vantaggio significativo nell'utilizzo di un algoritmo di gossip. Osservando l'andamento della Message Complexity è infatti chiaro come il crollo nel numero di messaggi transitati avvenga per EFF relativamente alti, cioè con probabilità di trasmissione vicine al 100%. Se invece si sceglie un valore di probabilità intermedio, si genera un numero di messaggi con un ordine di grandezza superiore al caso del puro flooding. Nel caso si utilizzi una cache di dimensione ancora superiore, la crescita nel numero di messaggi avviene sempre in modo lineare rispetto all'EFF ma risulta essere molto lenta, rendendo meno evidenti i vantaggi degli algoritmi di gossip rispetto al puro broadcast.
Per quanto rigarda il paragone tra il simulatore Lunes e la versione OmNET da noi implementata abbiamo riscontrato un comportamento simile nelle metriche relative alla diffusione di messaggi, risulato atteso in quanto si è cercato di comparare due simulatori che condividessero meccanismi simili. Se la message complexity e il Coverage hanno comportamenti qualitativamente e quantitativamente molto somiglianti, le metriche relative al tempo hanno andamenti simili ma la comparazione temporale pone Lunes in netto vantaggio. Ciò è probabilmente dovuto anche ad una scarsa ottimizzazione della nostra implementazione su OmNET.

Riferimenti bibliografici

[1] Ruijing Hu, Julien Sopena, Luciana Arantes, Pierre Sens, Isabelle Demeure

A fair comparison of gossip algorithms over large-scale random topologies,
Université Pierre et Marie Curie, CNRS/INRIA, Paris, France
Institut Telecom, Telecom ParisTech, CNRS, Paris, France
2012

[2] Gabriele D'Angelo, Stefano Ferretti

LUNES: Agent-based Simulation of P2P Systems,
Department of Computer Science, University of Bologna
Bologna, Italy
2011

[3] Matthew Szudzik, Wolfram Research, Inc.
An Elegant Pairing Function
NKS2006 Conference 2006