UNIVERSITÀ DEGLI STUDI DI PISA

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

TESI DI LAUREA

Availability-based persistence in reti sociali P2P

RELATORI

Prof.ssa Laura RICCI

Dott.ssa Barbara GUIDI

CANDIDATO

Francesco PITTO

Anno Accademico 2012 / 2013

Abstract

Le Distributed Online Social Networks (DOSNs) sono una sottoclasse delle OSNs che garantiscono un alto livello di autonomia e privacy. Questa tesi propone un approccio dinamico e totalmente distribuito al problema della persistenza dei dati nelle DOSNs che, basandosi sulla *availability* dei peer e sul concetto di *trust* tra utenti, garantisce un alto grado di disponibilità dei dati all'interno della rete. La persistenza dei dati nel sistema viene garantita attraverso l'elezione di un insieme di punti di memorizzazione, detti *social cache*, per i dati degli utenti. Le social cache, scelte in base alla loro centralità nella rete e alla loro disponibilità nel sistema, memorizzano o sono in grado di accedere ai dati dei propri amici offline, rendendoli disponibili a tutti i propri vicini che hanno diritto ad accedervi. È stata inoltre realizzata un'implementazione del nostro modello in PeerSim, un simulatore scalabile per reti P2P. I risultati raccolti con le nostre simulazioni testimoniano la bontà dell'approccio proposto.

Abstract

Distributed Online Social Networks (DOSNs) are a subset of OSNs that ensure a high level of autonomy and privacy. This thesis proposes a dynamic and fully distributed approach to the problem of data persistence in DOSNs. Our model is based on node *availability* and on *trust* between users and ensures a high level of data availability in the network. Data persistence in the system is achieved through the election of a set of points of storage, called *social caches*, for users' data. Social caches, which are chosen according to their centrality and their availability in the system, store or are able to access to their offline friends' data, making them available to all their neighbours who have the right to access them. We also show an implementation of our model in PeerSim, a scalable P2P simulator. The results obtained with our simulations show the validity of our approach.

Indice

Capitolo 1

Introduzione

Lo sviluppo su scala globale di Internet ha prodotto cambiamenti radicali non solo in tutte le discipline scientifiche e tecnologiche che hanno potuto beneficiare del suo utilizzo, ma soprattutto nelle relazioni sociali delle persone: Internet ha infatti consentito ai suoi utenti di mantenersi in contatto tra loro molto più semplicemente di quanto fosse possibile prima del suo avvento. Una delle qualità che hanno certamente favorito l'enorme diffusione della Rete è infatti la sua capacità di annullare le distanze e, di conseguenza, di riuscire a collegare in tempo reale persone che si trovano ai lati opposti del globo.

Internet ha progressivamente introdotto una serie di servizi, come ad esempio la posta elettronica, che hanno profondamente mutato il modo di comunicare e di interagire delle persone. Il servizio che più di ogni altro ha modificato, nell'ultimo decennio, il modo di relazionarsi degli utenti riguarda certamente le Online Social Networks (OSNs) [9]. Le OSNs sono piattaforme che permettono agli iscritti di costruire un proprio profilo e di stabilire connessioni tra esso e quello di altri utenti del sistema. Una connessione tra due utenti rappresenta un legame di qualche tipo fra essi, come ad esempio la volontà di scambiarsi informazioni, di interagire e di condividere i propri interessi. L'insieme dei legami tra gli utenti può essere modellato attraverso un grafo, dove i nodi rappresentano gli utenti e gli archi modellano le relazioni di interesse.

Le OSNs hanno avuto uno sviluppo eccezionale nell'ultimo decennio, tanto da occupare attualmente un'ampia porzione del tempo trascorso online da molti utenti. Proprio questa enorme e repentina diffusione delle più importanti OSNs (come Facebook, Twitter o Google+) in tutti i livelli della società ha concentrato su di esse l'attenzione sia delle grandi aziende di marketing sia della comunità scientifica. In particolare nelle social network è presente un'enorme mole di dati personali che gli utenti vorrebbero condividere solo con coloro con cui hanno stabilito autonomamente una relazione. Tuttavia tali

dati sono di grande interesse a diversi livelli e per molteplici scopi e sono forzatamente condivisi da ciascun utente con il fornitore del servizio, la cui politica di rispetto della privacy può non essere sempre del tutto limpida.

Proprio la crescente necessità di privacy e il maggior valore che l'utente percepisce nei confronti dei propri dati, ha spinto gli utenti delle OSNs a richiedere un maggior controllo dei propri dati. Tale richiesta non è stata sufficientemente soddisfatta dal provider dei servizi di OSNs centralizzate e ciò ha spinto il mondo accademico e quello della ricerca a studiare alternative alle OSNs che garantissero un maggior livello di privacy. Ci sono inoltre alcuni altri problemi presenti nelle OSNs a cui vorremmo dare una soluzione con le nuove proposte. Innanzitutto le OSNs hanno un problema di portabilità delle informazioni sociali: difficilmente infatti un utente riesce a portare i propri dati da una piattaforma all'altra. Tale operazione, essendo il servizio proprietario, risulta complessa e, soprattutto, poco conveniente per il fornitore del servizio, che tende quindi a non fornire interfacce adeguate. L'ultimo problema importante è quello della scalabilità: un approccio centralizzato comporta infatti un ampio costo di gestione e mantenimento delle infrastrutture della social network che, in caso di picchi di traffico straordinari, potrebbe fornire un servizio di qualità più bassa del normale.

La principale proposta su cui si stanno concentrando gran parte dei lavori di ricerca consiste nell'organizzare le OSNs in maniera distribuita: tale approccio ha portato alla definizione delle Distributed Online Social Networks (DOSNs) [14]. Secondo questa visione non esiste un'entità centrale che decide i termini del servizio offerto, ma sono i singoli nodi che garantiscono il corretto funzionamento del sistema cooperando tra loro secondo un approccio di tipo peer-to-peer. Ogni nodo, in questo modello, agisce sia da fornitore che da fruitore del servizio, decidendo le modalità con le quali memorizzare e operare sui propri dati e risolvendo in questo modo i problemi legati al controllo centralizzato dell'informazione.

La decentralizzazione delle funzionalità di una OSN dà origine a una serie di problemi direttamente connessi al nuovo approccio P2P adottato:

- gestione della dinamicità dei nodi a livello infrastrutturale e sociale;
- politica di memorizzazione delle informazioni sociali e problema della loro persistenza;
- verifiche di scalabilità e eventuale gestione di problemi di bilanciamento del carico;
- gestione dei *social update* che arrivano nel sistema;
- ricerca di nuovi utenti e problema della loro connessione nella rete;

- sicurezza dei dati.

Uno dei problemi più importanti da risolvere legati alla decentralizzazione di una OSN è quello della disponibilità e della persistenza dei dati. È necessario infatti che i dati di un utente siano sempre disponibili per gli utenti che hanno diritto ad accedervi: il compito risulta relativamente semplice quando l'utente proprietario dei dati si trova online, ma costituisce un problema ben più profondo durante i periodi in cui l'utente proprietario si trova offline.

Una delle soluzioni maggiormente utilizzate per il problema in esame è l'utilizzo di una Distributed Hash Table (DHT) [4] formata dagli utenti online in ogni momento. Tuttavia l'utilizzo di una simile struttura non consente all'utente proprietario dei dati di controllare dove essi saranno memorizzati. La nostra proposta si basa invece sull'utilizzo di social cache, server locali selezionati tra i nodi con cui il proprietario dei dati p ha stabilito una relazione, che si occupano di memorizzare i dati e di renderli disponibili per conto di p nei momenti in cui esso si trova offline. La nostra proposta introduce quindi un livello di *trust* nel sistema: vogliamo non solo garantire che i dati di un utente siano sempre accessibili, ma anche che essi vengano memorizzati da un utente verso cui il proprietario di dati ha una relazione di fiducia.

Lo scopo di questo lavoro di tesi è quindi la definizione di un modello, basato su un approccio egocentrico innovativo, per il problema della disponibilità e della persistenza dei dati in una DOSN. Il nostro obiettivo è quindi duplice: da un lato vogliamo individuare una serie di "potenziali" punti di memorizzazione nella rete per i dati dei diversi utenti. Allo stesso tempo vogliamo realizzare una sorta di "overlay dinamico" costituito dai punti di memorizzazione stessi insieme ad altri nodi, che garantisca la persistenza dell'informazione all'interno della rete assicurando il soddisfacimento dei vincoli di *trust* a cui si faceva cenno sopra.

Il nostro modello apre una serie di questioni e di problemi che nel corso della nostra tesi abbiamo affrontato e, in buona parte, risolto. In primo luogo è necessario definire quali sono le caratteristiche che un nodo deve avere per essere scelto come punto di memorizzazione dei dati di alcuni dei suoi vicini. La maggior parte degli studi sull'argomento si sono concentrati sull'individuazione di nodi caratterizzati da alta *centralità* nella rete, in modo da rendere le informazioni memorizzate disponibili a un alto numero di peer (i vicini nel grafo). Questa tesi ha invece come scopo principale la selezione di nodi sulla base della loro *disponibilità* all'interno del sistema. Vogliamo infatti che i nostri punti di memorizzazione siano quanto più possibile stabili e duraturi nel tempo, in modo da ridurre il traffico generato dalla ridistribuzione dei dati di un nodo che sta

per andare offline: in questo caso infatti, per garantire la persistenza dei dati, esso dovrà passare le informazioni memorizzate a nodi che sono ancora online.

Per individuare quali siano i fattori più adatti a descrivere il comportamento temporale di un peer all'interno di una OSN abbiamo condotto uno studio approfondito sull'*availability* dei nodi in tre diverse social network: Tagged, Netlog e MySpace [24]. Abbiamo cercato di comprendere in che misura il comportamento di un nodo fosse predicibile sulla base del suo comportamento passato, in modo da riuscire ad individuare in ogni momento quali peer hanno la maggior probabilità di rimanere ancora a lungo connessi nella rete. Il nostro studio ha mostrato come la durata media delle sessioni di un utente sia un buon indicatore per stimare il tempo che un peer online trascorrerà nel sistema prima di disconnettersi.

Abbiamo quindi proposto degli algoritmi distribuiti che riescono da un lato a eleggere nodi che costituiscono dei buoni punti di memorizzazione dell'informazione e dall'altro selezionano un insieme di nodi che assicurano che la persistenza dell'informazione all'interno della rete sia garantita con il soddisfacimento di un insieme di vincoli di *trust*. Uno dei principali contributi del nostro lavoro è poi l'introduzione di un algoritmo di rielezione dinamica dei nodi selezionati che tiene conto dell'evoluzione del sistema: abbiamo infatti formulato un algoritmo di rielezione che garantisce il soddisfacimento delle proprietà sopra elencate anche dopo il passaggio offline di alcuni dei punti di memorizzazione.

I contributi principali di questa tesi sono i seguenti:

- Uno studio dettagliato del comportamento temporale degli utenti all'interno delle OSNs. Lo studio comprende un'indagine sulla predicibilità della *availability* dei peer sulla base del loro comportamento passato e l'individuazione di una formula che stimi con un certo livello di attendibilità il tempo che un utente trascorrerà nel sistema prima di disconnettersi;

- Una strategia per l'elezione di un overlay dinamico che garantisca la persistenza dell'informazione in una DOSN con il soddisfacimento di alcuni vincoli di *trust*: in particolare vogliamo che un utente "dia in consegna" i propri dati solo ai propri vicini nel grafo sociale e che un utente che vuole accedere ai dati di un amico a offline lo possa fare solo attraverso un amico comune che egli ha con a;

- Una strategia distribuita di selezione di buoni punti di memorizzazione all'interno della rete che tenga conto, oltre alla centralità del nodo, delle sue caratteristiche di disponibilità;

- Un algoritmo distribuito per la rielezione dei nodi dell'overlay andati offline, in modo da garantire in ogni momento il più alto livello di persistenza dell'informazione possibile.

Il resto di questa tesi è organizzato come segue: nel Capitolo 2 viene fatto un quadro generale sullo stato dell'arte. Vengono descritte alcune proprietà generali delle reti complesse, per poi concentrarci più in dettaglio sulle DOSNs e sui problemi a esse relativi. Verrà quindi introdotto il problema del Social Caching, che ha costituito la più importante fonte di ispirazione per la nostra proposta. Discuteremo quindi alcuni modelli per la *availability* dei peer nelle OSNs e introdurremo infine le *temporal network*, un formalismo per la modellazione di grafi dinamici. Nel Capitolo 3 presenteremo in dettaglio il nostro approccio al problema del Social Caching e come intendiamo adattarlo al problema della persistenza dei dati. Verrà discussa una soluzione originale a tale problema, basata sui risultati ottenuti nell'analisi della disponibilità degli utenti in OSNs reali. Nel Capitolo 4 verrà mostrata l'implementazione del modello descritto nel capitolo precedente realizzata con PeerSim [38], un simulatore di reti P2P. Il Capitolo 5 sarà quindi dedicato alla descrizione dettagliata delle prove sperimentali condotte e ai risultati ottenuti. Nella prima parte viene riportata in dettaglio la nostra analisi sulla disponibilità degli utenti di alcune social network: i risultati di tale studio sono stati utilizzati nel Capitolo 3 per la selezione di buoni punti di memorizzazione per i dati dei nodi. Nella seconda parte mostreremo invece i risultati ottenuti con i nostri algoritmi sul modello implementato con PeerSim nel capitolo precedente: essi mostrano che il nostro approccio riesce a individuare come punti di memorizzazione peer caratterizzati da alta centralità e ottima disponibilità all'interno della rete e consente l'accesso ai dati di un utente offline da parte di tutti gli utenti che ne hanno diritto. Infine il Capitolo 6 contiene le considerazioni finali sul lavoro svolto e alcuni dei possibili sviluppi futuri della nostra proposta.

Capitolo 2

Stato dell'arte

Negli ultimi anni le Online Social Networks (OSNs) sono diventate uno dei servizi Internet più popolari. Le reti sociali possono essere modellate come reti complesse con caratteristiche particolari, in cui le interazioni tra gli utenti rivestono un ruolo di primaria importanza. Negli ultimi anni è emerso come la dinamicità degli utenti delle OSNs sia centrale per la progettazione di algoritmi per tutti i principali problemi su tali reti. In particolare si sta palesando la crescente necessità di avere strumenti che sintetizzino in maniera compatta l'evoluzione temporale di sistemi dinamici: sono stati proposti diversi modelli per tentare di raggiungere tale scopo.

In questo capitolo facciamo il punto sullo stato dell'arte in tutti questi ambiti che, integrati, costituiscono il fulcro di questa tesi. Nel Paragrafo 2.1 introdurremo brevemente le reti complesse, analizzandone le caratteristiche e le principali proprietà. Ci concentreremo quindi sulle OSNs, indagandone i limiti e spiegando perché sia emersa la necessità di proporre un nuovo modello di reti sociali basate su un approccio distribuito (Paragrafo 2.2). Le maggiori Distributed Online Social Networks (DOSNs) esistenti vengono presentate nel Paragrafo 2.3 e i principali problemi a esse relativi vengono discussi nel Paragrafo 2.4. Ci concentriamo poi sul problema del social caching, che sarà quello che analizzeremo più in dettaglio in questa tesi (Paragrafo 2.5). Sposteremo quindi la nostra attenzione sull'analisi della availability degli utenti nelle reti sociali (Paragrafo 2.6), mostrando come essa sia di assoluta rilevanza per il nostro problema. A questo scopo introdurremo infine le *temporal network* (Paragrafo 2.7), un formalismo recentemente proposto che consente di introdurre la dimensione temporale all'interno dei grafi: grazie ad esse abbiamo un modo per rappresentare il dinamismo di una rete sociale.

2.1 Reti complesse e loro proprietà

Quando pensiamo a una rete, o a un grafo, immaginiamo generalmente un insieme di nodi (o vertici) connessi da archi. Tuttavia le topologie di reti esistenti sono estremamente eterogenee, con caratteristiche peculiari che le distinguono le une dalle altre: le reti biologiche, Internet e le reti per la distribuzioni di varie risorse sono tutti esempi di reti che presentano, già a livello macroscopico, caratteristiche molto diverse le une dalle altre.

Lo studio dei grafi si è negli anni soprattutto concentrato sull'analisi di reti di piccole dimensioni, in cui è possibile estrarre caratteristiche interessanti per singoli vertici o singoli archi. Tuttavia, soprattutto nel corso dell'ultimo decennio, è emersa la crescente necessità di poter analizzare reti sempre più complesse. Queste sono reti che possono arrivare a contare anche alcuni milioni di nodi, per cui l'analisi delle caratteristiche dei singoli vertici risulta del tutto irrilevante. Siamo interessati invece a conoscere le caratteristiche complessive della rete, che ci permettono di comprenderne la struttura e di predirne l'evoluzione nel tempo. Si osserva inoltre che, data la loro complessità, tali reti non possono essere rappresentate graficamente: sono quindi assolutamente necessari un modello e un insieme di proprietà che ci permettono di intuire le caratteristiche della rete anche senza avere un modello grafico. Uno degli esempi più tipici di rete complessa sono le Online Social Networks (OSNs), reti composte da un insieme di persone o di gruppi di persone che interagiscono online tra loro. Le OSNs sono il principale oggetto di studio di questa tesi.

2.1.1 Clustering

Una delle principali proprietà che ci permette di classificare una rete complessa è il coefficiente di clustering, che misura in che grado i vicini di un nodo sono a loro volta vicini tra loro. In molte reti reali è stato osservato come esistano gruppi di nodi caratterizzati da un'altissima interconnessione reciproca.

Esistono diverse definizioni di coefficiente di clustering. Se consideriamo tale misura dal punto di vista di un singolo vertice v, essa misura il numero di archi esistenti tra i vicini del nodo rispetto al numero massimo di archi potenzialmente esistenti tra i vicini di v. Sia $n_v = |N(v)|$ il numero di vicini del nodo v. Il numero massimo di archi esistenti tra i vicini è dato da:

$$\binom{n_v}{2} = \frac{n_v(n_v - 1)}{2} \tag{2.1}$$

Dato un grafo non orientato $G = (V, E)$, se indichiamo con m_v il numero di archi del grafo indotto da $N(v)$, possiamo definire il coefficiente di clustering locale del nodo v come:

$$cc(v) = \begin{cases} m_v / \binom{n_v}{2} = \frac{2 \cdot m_v}{n_v(n_v-1)} & \text{if } \delta(v) > 1 \\ undefined & \text{altrimenti} \end{cases} \tag{2.2}$$

dove $\delta(v)$ indica il grado del vertice v. In modo simile si definisce il coefficiente di clustering locale per un vertice di un grafo orientato:

$$cc(v) = \begin{cases} m_v / (2 \cdot \binom{n_v}{2}) = \frac{m_v}{n_v(n_v-1)} & \text{if } \delta(v) > 1 \\ undefined & \text{altrimenti} \end{cases} \tag{2.3}$$

dove $\delta(v) = \delta_{in}(v) + \delta_{out}(v)$.

Il coefficiente di clustering globale ($CC(G)$) della rete può essere semplicemente ottenuto attraverso la media dei coefficienti di clustering dei singoli nodi per cui il valore del coefficiente di clustering non è indefinito:

$$CC(G) = \frac{1}{|V^*|} \sum_{v \in V^*} cc(v) \tag{2.4}$$

dove $V^* = \{v \in G |\ \delta(v) > 1\}$.

Tuttavia in [41] è stata proposta una definizione differente di coefficiente di clustering globale, chiamata comunemente *network transitivity*, che misura il rapporto tra il numero totale di triangoli distinti del grafo G ($n_\Delta(G)$) e il numero di triple distinte del grafo G ($n_\Lambda(G)$). Un triangolo di un grafo G è un sottografo indotto da una terna di vertici di G mutuamente adiacenti, mentre una tripla è una terna di vertici di G di cui almeno uno è adiacente a entrambi gli altri.

$$\tau(v) = \frac{n_\Delta(v)}{n_\Lambda(v)} \tag{2.5}$$

Per una social network il coefficiente di clustering di un grafo $G = (V, E)$ è detto *network density* ed è definito formalmente dall'equazione:

$$\rho(G) = \frac{|E|}{\binom{|V|}{2}} \tag{2.6}$$

In generale il coefficiente di clustering è compreso in $[0, 1]$, dove valori vicini a 0 indicano una rete poco clusterizzata e valori prossimi a 1 una rete altamente clusterizzata.

2.1.2 Degree distribution, modello scale-free e network resilience

Un'altra misura che caratterizza la struttura di una rete complessa è il grado dei vertici, o *vertex degree*, cioè il numero di archi incidenti a un nodo. Questa informazione ci fornisce indicazioni di fondamentale importanza sul ruolo dei nodi all'interno della rete: se molti gradi dei nodi risultano uguali significa che la struttura della rete è estremamente regolare, se al contrario ci sono nodi con grado molto maggiore della media la rete è caratterizzata dalla presenza di *hub*, nodi che fanno da tramite verso molti altri vertici della rete. Tali nodi sono di particolare rilevanza per la diffusione dell'informazione all'interno del sistema e la loro rimozione potrebbe provocare grossi danni al sistema, producendo un partizionamento della rete in diverse componenti.

Questa osservazione ci porta a discutere un'alta proprietà delle reti complesse nota come *network resilience*, ovvero la capacità della rete di mantenere un accettabile livello di servizio anche in seguito alla rimozione di alcuni nodi. In questo contesto si osservano capacità di resistenza diverse delle reti a seconda della modalità di scelta dei nodi da rimuovere: in particolare molte reti mostrano una buona resistenza a crash di nodi scelti in modi casuale, mentre hanno molte più difficoltà a mantenere alti standard di servizio nel caso si rimuovano i nodi con grado più alto della rete.

È interessante analizzare anche in che misura vertici con grado uguale o differente siano connessi tra loro. In molte reti reali è stato osservato come da un lato nodi con grado alto tendano ad essere connessi l'un l'altro, ma allo stesso tempo tendano ad essere connessi anche a nodi con gradi bassi [40]. Due misure per valutare tale correlazione sono l'*indice di correlazione di Pearson* e la *degree correlation*.

Sono stati effettuati numerosi studi sulla distribuzione dei gradi dei nodi in reti effettivamente esistenti. Barabàsi e Albert [5] hanno dimostrato come in molte reti reali il grado dei nodi segua una distribuzione di tipo power-law, sono cioè presenti un numero ridotto di nodi con grado alto e molti nodi con grado molto basso. Ciò mostra come i link all'interno di una rete non vengano creati con probabilità uniforme e che quindi il modello dei Grafi Random proposto da Erdős e Rényi in [18] non sia adatto a modellare le reti sociali. Una delle principali spiegazioni di tale fenomeno è il fatto che, in una rete che evolve dinamicamente, i nuovi nodi tendono a collegarsi preferibilmente a vertici già connessi ad un alto numero di nodi della rete. Tali reti vengono chiamate *scale-free*, dato che il grafo della distribuzione dei gradi dei nodi è di tipo esponenziale negativo, e quindi invariante per cambiamenti di scala.

2.1.3 Centralità

Alcune tra le misure più utilizzate per caratterizzare l'importanza di un nodo in una rete sono gli indici di centralità. A seconda del contesto in cui stiamo lavorando sono stati introdotti diversi indici di centralità, tra i quali i più usati sono la degree centrality (DC), la closeness centrality (CC) e la betweenness centrality (BC).

La Degree Centrality fornisce una misura del numero di archi posseduti da un nodo e misura la connettività di un nodo della rete. Essa è definita formalmente come segue:

$$DC(v) = \frac{\delta(v)}{|V| - 1} \tag{2.7}$$

dove $\delta(v)$ indica il grado del nodo v.

La Closeness Centrality calcola la media delle distanze di un nodo dagli altri. Fornisce una misura della rapidità di propagazione dell'informazione da un vertice verso tutti gli altri nodi della rete. Essa viene calcolata con la formula:

$$CC(v) = \frac{\sum_{t \in V} d(v,t)}{|V| - 1} \tag{2.8}$$

dove $d(v,t)$ rappresenta la distanza minima tra il nodo v e t.

La Betweenness Centrality di un nodo v è una misura della frazione di cammini minimi $\sigma_{s,t}(v)$ tra ogni coppia di nodi s e t che passano attraverso il nodo v:

$$BC(v) = \sum_{v \neq s,t} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}} \tag{2.9}$$

dove $\sigma_{s,t}$ è il numero di cammini minimi dal nodo s al nodo t. Tale misura ci fornisce una misura di quanto un peer ha il controllo dell'informazione che passa tra gli altri nodi. Purtroppo il suo costo computazionale risulta essere estremamente elevato per reti molto grandi.

2.1.4 Small-world

L'ultima proprietà delle reti complesse che vogliamo discutere in questo capitolo è quella di small-world: essa afferma che il diametro di una rete sociale è logaritmico nella dimensione della rete. Questo indica che il cammino minimo tra due nodi della rete è estremamente breve. La prima dimostrazione di tale fenomeno fu fornita nel 1960 dal sociologo Stanley Milgram con un noto esperimento: egli chiese a circa 160 persone scelte casualmente di consegnare una lettera a un operatore di borsa di Boston di cui era noto soltanto il nome. Ogni persona poteva consegnare la lettera solo a persone conosciute

che riteneva avere qualche punto di contatto con il destinatario della lettera. Milgram misurò che il numero di "passaggi di mano" di ogni lettera arrivata a destinazione era al più 6. Tale risultato, noto come teoria dei 6 gradi di separazione, mostra come ogni coppia di nodi in una rete sociale è collegata da un cammino caratterizzato da un numero limitato di hop e come alcuni modelli, come ad esempio le social network con struttura *grid-like*, sovrastimino la misura del diametro della rete.

2.2 Dalle OSNs alle DOSNs

La maggior parte delle attuali Online Social Network (OSNs) sono servizi web eseguiti su infrastrutture logicamente centralizzate [14]. Le OSNs forniscono una piattaforma online che

- offre servizi all'utente per la costruzione di un profilo pubblico e per stabilire esplicitamente connessioni con i profili degli altri utenti;
- permette all'utente di condividere informazioni e contenuti con utenti selezionati;
- supporta lo sviluppo e l'utilizzo di applicazioni sociali con cui l'utente può interagire e collaborare sia con amici che con sconosciuti.

Tuttavia le social network possiedono anche una serie di svantaggi: innanzitutto con la crescita esponenziale del numero degli utenti che vi si connettono possono soffrire di problemi di scalabilità. Inoltre la presenza di un unico provider centralizzato costituisce sia un collo di bottiglia per l'intera applicazione che un unico punto di fallimento. Oltre a ciò è da sottolineare come l'architettura centralizzata, comportando una certa mancanza di località, porti alla necessità di implementare opportune strategie per il mantenimento della privacy. Per tutti questi motivi negli ultimi anni sono state proposte numerose piattaforme che offrono all'utente i servizi supportati da una OSN in maniera distribuita: esse sono state denominate Distributed Online Social Networks (DOSNs). Le diverse DOSNs differiscono sostanzialmente per il modo con cui viene realizzata la decentralizzazione dei servizi offerti dall'OSN, che vanno dall'utilizzo di server decentralizzati, a quello di un sistema P2P, fino a sistemi ibridi che combinano i due approcci precedenti.

2.3 DOSNs: proposte esistenti

Presentiamo ora le caratteristiche di alcune delle principali DOSNs esistenti:

Diaspora Diaspora[1] è una DOSN realizzata attraverso una rete di server decentralizzati confederati che vengono amministrati dai singoli utenti: essi possono in questo modo gestire individualmente i propri dati. Alternativamente gli utenti possono aggregarsi ad un server già esistente dichiarando implicitamente la loro fiducia nel server a cui si aggregano. La comunicazione avviene attraverso il meccanismo dei post, la cui visibilità può essere pubblica, ovvero consentita a tutti, o limitata, cioè ristretta a uno specifico gruppo di utenti (*sharing group*) che può essere definito dall'utente stesso. Il meccanismo di notifica si basa su una strategia di tipo *push* ed è previsto inoltre un meccanismo di replicazione dei dati.

PeerSon PeerSon [10] è un'infrastruttura distribuita basata su un sistema P2P che offre un servizio di cifratura e scambio diretto di dati. Il sistema è basato su un'architettura di tipo $two-tier$ con un livello che offre un servizio di look-up realizzato attraverso una DHT e un secondo livello che contiene i dati dell'utente. Lo scambio diretto dei dati permette agli utenti di utilizzare il sistema anche in assenza di una connettività Internet persistente attraverso un sistema di messaggistica asincrona. Il sistema prevede anche la presenza di un *muro*, che contiene oggetti e notifiche postati dall'utente stesso o da suoi amici, e offre un servizio di trasferimento di file.

Safebook Safebook [13], anch'esso basato su un sistema P2P, è composto da tre componenti principali:

- un Trusted Identity Service (TIS), un servizio offline che permette agli utente di acquisire certificati con cui potersi unire alla rete;
- le matrioska, strutture concentriche che offrono un servizio di memorizzazione dei dati e garantiscono la privacy di ciascun nodo;
- un substrato basato su Kademlia [35] che offre il servizio di look-up.

All'interno del sistema si distinguono utenti certificati e utenti anonimi.

SuperNova SuperNova [44] è un sistema che si distingue nel panorama delle DOSNs per la sua architettura basata su Super-Peer. Questi sono nodi che per le loro caratteristiche (alta banda, ampio spazio di memorizzazione, più alta frequenza online rispetto agli altri nodi, etc.) possono aiutare altri utenti a connettersi e possono mantenere informazioni su di loro. In questo modo i nodi che acquisiscono lo stato di super-peer

[1] https://joindiaspora.com

costituiscono una sorta di piccolo indice locale da cui gli altri nodi possono acquisire informazioni per trovare altri peer con comportamento simile al loro. Nel sistema è presente anche un meccanismo di replicazione dei dati e un meccanismo di *timestamps* per gestire eventuali conflitti tra gli update.

LifeSocial.KOM LifeSocial.KOM [23] è una OSN basata su piattaforma P2P in cui l'overlay sottostante è FreePastry[2]. I dati vengono memorizzati in una DHT in maniera sicura e con un meccanismo che garantisce un buon bilanciamento del carico. Vengono garantite confidenzialità e sicurezza grazie anche ad un servizio di cifratura.

Confidant Citiamo infine Confidant [32] come esempio di approccio ibrido: i dati sono mantenuti a livello di rete P2P e sono replicati su nodi con i quali c'è una relazione di fiducia, ma altre informazioni rilevanti per l'utente (indirizzo dei nodi che possiedono le repliche dei suoi dati, lo stato di tali nodi, etc.) vengono invece mantenuti su dei server cloud.

2.4 Problemi aperti sulle DOSNs

In questo paragrafo presenteremo brevemente alcuni dei principali problemi di ricerca aperti relativi alle DOSNs ovvero quello della predizione dei contatti, quello della diffusione delle informazioni e quello della disponibilità dei dati. Concluderemo quindi con una breve sezione sull'importanza di avere algoritmi distribuiti per l'analisi di una rete sociale.

2.4.1 Link Prediction

Questo problema consiste nel prevedere quali interazioni tra i membri facenti parte della rete sociale è più probabile si verifichino nell'immediato futuro. Individuare peer che si trovano contemporaneamente online è un problema di grande interesse nell'ambito delle social network e permette ad esempio di gestire i dati e le loro repliche in modo intelligente. In questo campo sono stati effettuati numerosi studi nell'area dei P2P reccomended systems in cui si è tentato di classificare i diversi profili in base a note misure di similarità. Sono stati proposti diversi algoritmi basati sull'osservazione che i gruppi presenti nelle reti sociali tendono a espandersi attraverso amici di amici. Sono stati inoltre proposti algoritmi basati sulla popolarità di un peer e altri basati su misure di prossimità per cercare di individuare peer con comportamenti simili.

[2] http://www.freepastry.org/freepastry

2.4.2 Disponibilità dei dati

Uno dei principali problemi nelle DOSNs è garantire che i dati di un peer siano disponibili anche quando questo non si trova online. Tali dati contengono tutte le informazioni prodotte dal peer durante la sua attività online e devono essere rese disponibili ai suoi amici. Le tecniche maggiormente utilizzate per raggiungere tale obiettivo sono quelle del Distributed Caching [28] e del Dynamic Data Repication [1]: è in generale necessario garantire che più copie dei dati siano disponibili nella rete perché le informazioni siano accessibili in ogni momento. Tuttavia il problema di quante copie produrre per ogni informazione e a chi distribuirle è una questione al centro di numerosi articoli di ricerca [43].

2.4.3 Diffusione delle informazioni

Le attuali OSNs producono un'enorme quantità di informazioni. La loro diffusione nella rete deve essere un'attività coordinata, in modo da ridurre la duplicazione dei dati e il traffico nella rete e garantire agli utenti informazioni aggiornate e affidabili. Le due tecniche con cui le informazioni vengono diffuse nella rete sono essenzialmente basate su approcci multicast e algoritmi di gossip per la diffusione delle informazioni all'interno di specifiche comunità. L'approccio multicast garantisce una maggior velocità di diffusione dell'informazione all'interno della rete ma un approccio gossip produce un minor numero di messaggi. L'obiettivo complessivo è quello di garantire che gli aggiornamenti di un profilo siano disponibili quasi immediatamente per gli amici online, senza tuttavia produrre un'eccessiva quantità di traffico nella rete.

In [36] è proposto un protocollo efficiente di distribuzione dell'informazione basato sul gossip. Esso utilizza le storie dei messaggi e utilizza una complessa euristica di selezione del vicino a cui diffondere l'informazione che combina l'approccio random, l'utilizzo di una misura di anticentralità e la conoscenza delle componenti connesse del grafo. Sono quindi proposti due protocolli gossip per la propagazione degli update, uno di tipo *rumor mongering* con una strategia *push* e uno di tipo *anti-entropy* con strategia *push-pull*. Ad ogni ciclo di gossip vengono inviati tutti o una parte degli aggiornamenti che il vicino selezionato potrebbe non conoscere: tali informazioni vengono quindi rimosse dalla lista delle informazioni da diffondere con una certa probabilità p. Quando un nodo viene a conoscenza di un nuovo update per un suo contatto lo diffonde anche ai vicini in comune con tale nodo. Le storie dei messaggi vengono infine utilizzate per ridurre la probabilità che vengano inviati nuovamente aggiornamenti di cui il destinatario è già a conoscenza.

Un approccio particolare per ridurre il costo della diffusione dell'informazione è quello del Social Caching, consistente nell'eleggere un certo numero di nodi come social cache per fare da ponte tra i produttori e i consumatori dei social update presenti nella rete. Questo approccio riduce drasticamente il numero di connessioni necessarie per la diffusione delle informazioni all'interno della rete e di conseguenza il traffico di messaggi all'interno della rete stessa. Dato che il Social Caching costituisce la maggiore fonte di ispirazione per la proposta presentata in questa tesi, il problema viene esaminato in modo più approfondito nel Paragrafo 2.5.

2.4.4 Algoritmi distribuiti per l'analisi di reti sociali

Per misurare l'importanza di un nodo all'interno di una rete sociale sono stati proposti diversi indici di centralità. I più importanti, come discusso nel Paragrafo 2.1.3, sono la betweenness centrality (BC), la closeness centrality (CC) e la degree centrality (DC). Queste misure sono state adattate per il modello delle ego network, che verrà presentato nel Paragrafo 3.1: l'indice maggiormente utilizzato in tale modello è quello della ego-betweenness centrality (EgoBC).

I principali algoritmi utilizzati per il calcolo di tali indici, oltre a presentare generalmente un elevato costo computazionale, richiedono di avere una conoscenza globale della rete. Questi due fattori li rendono praticamente inutilizzabili nelle social network, in cui generalmente abbiamo solo una visione locale della rete e sono presenti centinaia di migliaia di nodi. Solo negli ultimi anni sono stati proposti una serie di algoritmi decentralizzati per il calcolo di tali indici o, nella maggior parte dei casi, di una loro approssimazione.

In DANCE [47] si propone un metodo per la stima della closeness centrality di un peer p che considera la sottorete dei nodi che si trovano a una distanza minore o uguale di una certa soglia h da p e utilizza messaggi con un *time-to-live* limitato.

In [11] si considerano solo reti modulari, in cui diverse comunità sono connesse tra loro. L'articolo propone un metodo che calcola un'approssimazione della closeness-centrality in $O(m)$, dove m è il numero di archi del social graph. Vengono introdotte inoltre due misure di centralità chiamate *Inbetweenness Centrality* e *Shortest-path Betweenness*.

In [20] vengono proposte tre diverse tecniche per l'approssimazione della betweenness centrality basate sulla selezione di un particolare sottoinsieme di nodi detti pivot.

Infine in [31] si propone un framework per il calcolo di quattro misure di centralità (Betweenness Centrality, Closeness Centrality, Graph Centrality e Stress Centrality) sot-

to ipotesi piuttosto restrittive: si assume infatti che il sistema possa essere sincronizzato e che ogni dispositivo abbia un raggio massimo di trasmissione. Il metodo è composto da due fasi denominate *Count Phase* e *Report Phase*: nella prima si calcolano la distanza o il numero di cammini verso tutti gli altri nodi della rete, nella seconda ogni nodo diffonde la misura calcolata agli altri nodi. La complessità temporale di tale metodi risulta essere nell'ordine del diametro della rete.

2.5 Il problema del Social Caching

In questa sezione affrontiamo in dettaglio il problema del Social Caching e mostriamo alcuni dei principali algoritmi proposti per la sua soluzione sotto diverse ipotesi. L'utilizzo delle social cache può costituire una soluzione a due diversi problemi. Nella versione originale, proposta in [26], le social cache vengono utilizzate per ridurre il costo della distribuzione delle informazioni all'interno della rete: esse costituiscono di fatto dei ponti per la diffusione degli update nel grafo sociale. Ogni social cache infatti memorizza gli update dei nodi a essa collegati e li rende disponibili agli utenti che ne hanno diritto. In questo modo, se il numero di social cache è inferiore a quello degli amici, un peer può conoscere gli aggiornamenti dei vicini senza doverli contattare singolarmente: sarà sufficiente infatti interrogare le proprie social cache per ricevere, con un unico messaggio, gli update di tutti gli amici a essa collegati. In questo modo si riescono a ridurre rispetto a un approccio push-pull completo sia il numero di connessioni che il numero di messaggi inviati nella rete. Per questa ragione le social cache vengono definite dei "bridge sociali" : esse, venendo utilizzate sia dai produttori che dai consumatori dell'informazione, mettono in contatto peer che sono amici nella rete sociale e che si trovano quindi al più a un hop di distanza. Per garantire la copertura di ogni arco della rete si deve fare in modo che ogni coppia di peer in cui nessuno dei due nodi è social cache abbia almeno un amico comune che è social cache.

Le social cache, come proposto in [15], possono anche essere usate per il problema della persistenza dei dati all'interno della rete. In questo contesto esse rendono disponibili agli utenti che ne hanno diritto i dati dei peer offline: ogni nodo all'interno della rete seleziona uno o più dei vicini (detti appunto social cache) che si occupano di memorizzare e rendere disponibili i suoi dati anche quando esso non si trova online. Si noti che, anche in questo caso, ogni social cache può contenere soltanto i dati dei nodi a esso connessi.

Vediamo ora un esempio concreto, che si riferisce all'approccio del Social Caching per la diffusione delle informazioni nella rete. Consideriamo il grafo mostrato in Figura 2.1(a) e una possibile selezione di social cache (mostrata in Figura 2.1(b)). Il nodo

3 è stato selezionato come social cache dei nodi {2,3,4}: tali nodi notificano quindi al nodo 3 i propri social update e lo interrogano quando vogliono apprendere i social update di uno degli altri nodi nell'insieme. Il nodo 5 si comporta allo stesso modo con i nodi {1,2,4,5}. Con questa selezione risulta che ogni coppia di nodi tra cui esiste una relazione di amicizia ha una social cache da cui apprendere e a cui notificare i social update: questo può essere uno dei due nodi tra cui esiste la relazione di amicizia oppure un vicino comune dei due nodi. Quest'ultimo caso è quello ad esempio dei nodi 1 e 2: nessuno dei due è una social cache, ma esiste un vicino comune che lo è (il nodo 5).

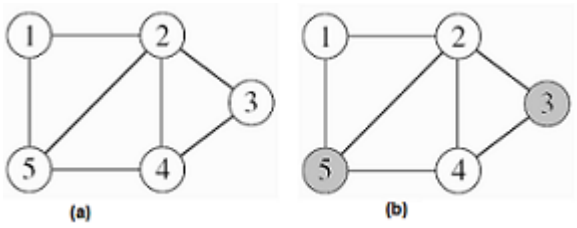


Figura 2.1: Selezione social cache

Questo semplice esempio ci permette di formalizzare più rigorosamente i requisiti che devono essere soddisfatti dai nodi selezionati come social cache. Si consideri il grafo non orientato $G = (V, E)$ dove V è l'insieme dei vertici e E l'insieme degli archi. Le social cache devono formare un *Neighbor-Dominating Set (NDS)* ovvero un insieme di vertici tale che per ogni arco del grafo uno dei due nodi incidenti è nel neighbor-dominating set oppure vi è uno dei vicini comuni. Più formalmente vogliamo un insieme $S \subseteq V$ tale che $\forall (u, v) \in E$ esiste un $w \in S$ che soddisfa $w \in (N(u) \cap N(v)) \cup \{u, v\}$. Un arco per cui vale la proprietà precedente è detto *coperto*. Collegato a questo concetto è quello dell'NDS-problem, ovvero quello di trovare, dato un grafo non orientato $G = (V, E)$, un neighbor-dominating set di dimensione minima. Nel caso precedente si osserva ad esempio che il nodo 2 può essere utilizzato come social cache per tutti gli archi e che quindi costituisce da solo il neighbor-dominating set di dimensione minima.

La scelta dei nodi da selezionare come social cache può essere fatta sulla base di criteri diversi, ma che tengono generalmente conto delle proprietà locali e globali del nodo. Nel seguito analizzeremo in dettaglio alcuni dei principali algoritmi che risolvono il problema sotto diverse ipotesi assuntive.

2.5.1 Social Butterfly

In [25] vengono proposti due algoritmi che risolvono il problema sotto l'ipotesi aggiuntiva di conoscere l'intera topologia della rete sociale. L'obiettivo è quello di selezionare il numero più piccolo possibile di social cache.

Approximate NDS Algorithm Questo algoritmo genera per ogni vertice $v_i \in V$ un insieme S_i che contiene un insieme di archi $e = (\alpha, \beta)$ in cui $v_i \in \{\alpha, \beta\}$ o v_i è un vicino comune di (α, β). In pratica S_i è l'insieme degli archi che sarebbero coperti da una eventuale elezione di v_i come social cache. L'algoritmo procede quindi in modo greedy, eleggendo come social cache ad ogni passo il vertice v_i il cui corrispondente insieme S_i contiene il più grande numero di archi non ancora coperti da una social cache. L'algoritmo, mostrato in Algorithm 1, termina quando tutti gli archi sono coperti da una social cache.

Algorithm 1 Approximate NDS Algorithm

Input: undirected graph $G = (V, E)$
Output: Social Caches C
Initialization:
Universal Set = E;
Covered Set = $\varnothing$;
for each vertex $v_i \in V$ **do**
    generate S_i contains edges $e = (\alpha, \beta)$, where $v_i \in (N(\alpha) \cap N(\beta)) \cup \{\alpha, \beta\}$;
end for
Main Algorithm:
while Universal Set != $\varnothing$ **do**
    select S_i that maximizes |Universal Set $\cap$ S_i|;
    Universal Set = Universal Set - S_i;
    Covered Set = Covered Set $\cup$ S_i;
end while

Social Score Algorithm Questo algoritmo seleziona i nodi che diventeranno social cache in base alle caratteristiche locali e di connettività che essi hanno all'interno della rete. Ad ogni nodo viene attribuito un punteggio di Social Score che misura la capacità del nodo di svolgere il ruolo di social cache ed è calcolato sulla base di tre diversi fattori:

- il *coefficiente di clustering (CC)*, che è una misura di quanto i vertici di un grafo tendono ad aggregarsi in cluster;

- l'*egocentric betweennes centrality (EBC)*, che è una misura di quanto un vertice può facilitare la comunicazione tra altri nodi all'interno della rete;
- il *grado di un nodo (D)*, ovvero il numero dei suoi vicini.

Il Social Score di ogni nodo viene calcolato come una combinazione del coefficiente di clustering e dell'egocentric betweennes centrality scalato in base al grado del nodo:

$$SS = ((1 - CC) + EBC) \cdot D \tag{2.10}$$

L'algoritmo mantiene per ogni vertice una tabella che contiene, oltre alle tre misure elencate sopra, un campo booleano $IsCache$ che indica se il vertice è una social cache o meno, un campo $CacheList$ che indica le cache a cui il nodo è affiliato e un campo $MemberList$ che contiene tutti i membri all'interno del cluster sociale nel caso il nodo sia una cache. L'algoritmo, mostrato in Algorithm 2, dopo una fase di inizializzazione, seleziona ad ogni passo il vertice con il valore più alto di Social Score e considera tutti i suoi vicini non ancora inseriti nel cluster di altri nodi come membri del social cluster di tale nodo.

2.5.2 SocialCDN

In [26] gli autori propongono un sistema di distribuzione delle informazioni all'interno di una rete sociale basato su un meccanismo distribuito di selezione delle social cache. Vengono proposti quattro algoritmi distribuiti per la selezione delle social cache che non richiedono la conoscenza completa del grafo sociale come avveniva invece in [25]. Tali algoritmi sono *Randomized Algorithm*, *Triad Elimination Algorithm*, *Span Elimination Algorithm* e *Social Score Algorithm*. Anche in questo caso l'obiettivo è quello di selezionare un NDS di dimensione minima.

Notazione:

- $deg(v)$ è il grado del nodo v;
- $N(v)$ è l'insieme dei vicini del nodo v;
- S_v è l'insieme degli archi coperti dal nodo v ovvero ogni arco $e = (\alpha, \beta) \in E$ tale che $v \in (N(\alpha) \cap N(\beta)) \cup \{\alpha, \beta\}$;
- $T(v)$ è il numero di *triadi transitive* del nodo v. Una *triade transitiva* è un insieme di tre nodi costituito da tre diadi che possiedono anche la proprietà della

Algorithm 2 Social Score Algorithm

Input: undirected graph $G = (V, E)$
Outputs: Social Caches C and Social Clusters SC
Initialization:
for each vertex $v_i \in V$ **do**
    calculate CC_i, EBC_i, SS_i;
    set $IsCache_i$, $CacheList_i$, $MemberList_i$ to $\varnothing$;
end for
Main Algorithm:
for each vertex in V **do**
    pick vertex v_j with highest Social Score;
    if $CacheList_j == \varnothing$ **then**
        $IsCache_j = true$;
        $MemberList_j = v_j$'s friends F_j;
    else
        get $CacheList_j$ and friends of $v_j : F_j$;
        for each cache $C_k \in CacheList_j$ **do**
            get friends of $C_k : F_k$;
            $MemberList_j = F_j - F_k$;
        end for
        if $MemberList_j$!= $\varnothing$ **then**
            $IsCache_j = true$;
            update $MemberList_j$;
        end if
    end if
end for

transitività, ovvero se ogni volta che esiste una relazione sociale (quindi un arco nel social graph) tra il nodo A e il nodo B e una relazione sociale tra il nodo B e il nodo C allora esiste anche una relazione sociale tra il nodo A e il nodo C;

- $CN(u, v)$ indica i vicini comuni tra u e v e $CN(e)$ indica i vicini comuni tra i nodi incidenti all'arco e;
- ogni nodo v è etichettato come *Black* se è stato selezionato come social cache, *Grey* se ogni arco in S_v è coperto da una social cache ma v non è stato selezionato come social cache o $White$ se v non è una social cache e c'è almeno un arco in S_v non coperto;
- gli archi coperti da una social cache sono etichettati come $Green$, quelli non coperti come *Red*;
- $span(v)$ è il numero di archi *red* in S_v.

Randomized Algorithm Questo algoritmo viene utilizzato soprattutto come standard di riferimento per la valutazione e il confronto degli altri tre. Ogni nodo v esegue i seguenti passi:

1. calcola $span(v)$;
2. se $span(v) = 0$ (tutti gli archi in S_v sono etichettati come $Green$) il nodo v è etichettato come $Grey$ e termina;
3. se $span(v) > 0$ genera randomicamente un numero $p(v) \in [0, 1]$: se tale valore è maggiore di una soglia prefissata θ il nodo diventa social cache, viene etichettato come *Black*, etichetta come $Green$ tutti gli archi in S_v, informa i vicini della propria elezione e termina. Se invece $p(v) < \theta$ non fa niente.

   Quando l'algoritmo termina le proprietà del NDS sono rispettate; le sue performance dipendono dalla probabilità soglia θ.

Triad Elimination Algorithm Questo algoritmo, come anche lo *Span Elimination Algorithm*, è costituito da due fasi. Nella *selection phase* del *Triad Elimination Algorithm* ogni nodo calcola il numero di triadi transitive di cui fa parte, per il cui calcolo è sufficiente conoscere i vicini a 1 e 2 hop di distanza. Quindi per ogni arco $e = (u, v)$, u e v si scambiano i valori di $T(u)$ e $T(v)$ e il nodo tra i due che ha il valore maggiore viene eletto come *Temporary Social Cache* dell'arco ($TSC(e)$). La fase successiva, detta

elimination phase, ha il compito di ridurre il numero di cache selezionate: al termine della *selection phase* infatti tutti gli archi sono certamente coperti (sono cioè *Green*) ma il numero di social cache selezionate non è ottimo. L'*elimination phase* si basa sull'osservazione che ogni vicino comune dei nodi incidenti su un arco, può essere utilizzato come social cache per quell'arco. In questa fase per ogni arco e la $TSC(e)$ contatta tutti i vicini comuni di $e = (u, v)$ e elegge quello che è stato selezionato il maggior numero di volte nella fase precedente come social cache definitiva per quell'arco. Più precisamente ogni arco $e = (u, v)$ confronta il numero di volte che è stato selezionato $freq(TSC(e))$ con $freq(w)$ per ogni $w \in CN(u, v)$. Il nodo $n \in (N(u) \cap N(v)) \cup \{u, v\}$ con il più alto valore $freq(n)$ verrà selezionato come cache per quell'arco: il nodo n verrà etichettato come *Black*, marcherà tutti gli archi in S_v come *Green*, informerà i vicini della sua elezione e terminerà quindi l'esecuzione dell'algoritmo. L'algoritmo termina in un numero costante di round pari a 2.

Span Elimination Algorithm Questo algoritmo è molto simile al precedente. La differenza consiste nel fatto che nella *selection phase* invece di calcolare e scambiare su ogni arco il numero di triadi transitive, viene calcolato il numero di archi che ogni nodo v copre, ovvero il valore $size(S_v)$. Il comportamento della parte restante dell'algoritmo è del tutto identica a quello del *Triad Elimination Algorithm*: viene eletta una $TSC(e)$ per ogni arco e e si procede quindi alla *elimination phase*, che è identica a quella dell'algoritmo precedente. L'algoritmo termina in un numero costante di round pari a 2.

Social Score Algorithm In questo algoritmo l'elezione delle social cache si basa sulla misura di Social Score descritta nel Paragrafo 2.5.1. Anche questo algoritmo opera in due fasi: nella prima ogni nodo v calcola il proprio valore di Social Score ($ss(v)$) utilizzando l'Equazione 2.10. Da tale valore si calcola $ssprob(v)$ utilizzando l'Equazione 2.11:

$$ssprob(v) = \begin{cases} 1 - 1/ss(v) \text{ se } ss(v) \geq 1 \\ 0 \text{ se } ss(v) < 1 \end{cases} \tag{2.11}$$

Questo valore esprime la probabilità che un nodo diventi social cache. Ciò accade se $ssprob(v)$ supera una certa soglia ρ: in questo caso viene etichettato come *Black*, etichetta come *Green* tutti gli archi in S_v, informa i vicini della propria elezione e termina.

Nella seconda fase tutti i nodi che non sono stati eletti social cache calcolano il valore $span(v)$. Se tale valore è pari a 0 (tutti gli archi in S_v sono coperti) il nodo diventa *Grey*

e termina, altrimenti calcola il valore $ratio(v)$ utilizzando l'Equazione 2.12 e se tale valore è maggiore di una certa soglia γ il nodo diventa social cache, viene etichettato come *Black*, etichetta come *Green* tutti gli archi in S_v, informa i vicini della propria elezione e termina. La probabilità γ diminuisce ad ogni ciclo.

$$ratio(v) = \frac{span(v)}{size(S_v)} \tag{2.12}$$

Dato che un adattamento di questo algoritmo verrà utilizzato nel seguito mostriamo lo pseudocodice della Fase 1 in Algorithm 5 e quello della fase 2 in Algorithm 6.

Le performance di questo algoritmo dipendono dal valore delle probabilità soglia scelte, ma termina in $1+\frac{1}{RATIOSTEPSIZE}$ round, dove $RATIOSTEPSIZE$ è la quantità di cui si decrementa ad ogni passo il parametro γ.

Algorithm 3 Social Score Algorithm - Fase 1

```
calculate ssprob(v);
if ssprob(v) > ρ then
    mark itself as Black (*social cache*);
    mark edges in SS_v as Green;
    inform every node in N(v);
end if
```

Algorithm 4 Social Score Algorithm - Fase 2

```
while span(v) > 0 do
    calculate ratio(v);
    if ratio(v) > γ then
        mark v as Black (*social cache*);
        mark red edges in SS_v as Green;
        inform every node in N(v);
    else
        γ− = RATIOSTEPSIZE;
        recalculate span(v);
    end if
end while
```

2.5.3 Social cache per la persistenza dell'informazione

In [15] viene proposto un primo algoritmo per la selezione di social cache con l'obiettivo della persistenza dell'informazione. In questo approccio le social cache costituiscono dei potenziali punti di memorizzazione dei dati dei nodi e permettono ai propri vicini che

ne hanno diritto di accedervi anche quando il proprietario delle informazioni si trova offline. Il modello è basato sul alcuni vincoli di *trust*: ogni nodo vuole che le proprie informazioni sociali siano date in consegna nei periodi in cui è offline solo ai propri vicini della rete e vuole accedere ai dati di un vicino offline solo attraverso un vicino comune che ha con esso. La social overlay è invece basata sul concetto di ego network che può essere semplicisticamente definita come la rete che un nodo costituisce con i 150 amici con cui ha contatti più frequentemente. Una definizione più dettagliata di ego network verrà fornita nel Paragrafo 3.1.

L'algoritmo è ispirato a quello di Social Score proposto in [26] e presentato nel Paragrafo 2.5.2. Esso viene eseguito su una configurazione del grafo sociale in cui tutti i nodi si trovano online e si suppone che i nodi che vengono eletti social cache non vadano mai offline, garantendo in questo modo il soddisfacimento dei vincoli di *trust* in ogni momento. L'algoritmo seleziona inizialmente un insieme di social cache nella rete sulla base di un punteggio di Social Score che è tanto più alto per un nodo tanto più esso è adatto a svolgere il ruolo di social cache. Il Social Score è calcolato per ogni nodo della rete secondo una delle seguenti formule:

$$SS1 = DynamicEBC + NodeAvailability \tag{2.13}$$

$$SS2 = DynamicWEBC + NodeAvailability \tag{2.14}$$

L'obiettivo di tali formule è racchiudere in una misura unica informazioni sulle caratteristiche sia di centralità che temporali di ogni nodo: la $DynamicEBC$ e la $DynamicWEBC$ sono infatti misure di centralità del nodo all'interno della rete mentre la $NodeAvailability$, definita come $NodeAvailability = \frac{TempoMedioOnline}{TempoMedioOffline}$ è una misura della frazione di tempo trascorso online dal peer. Un nodo con una $NodeAvailability$ alta trascorre più tempo online nel sistema e garantisce quindi una maggiore disponibilità dei dati.

Presentiamo ora brevemente l'algoritmo proposto in [15] per l'elezione delle social cache. Durante l'algoritmo di selezione ogni nodo v è etichettato come *Black* se è stato selezionato come social cache, *Grey* se non è stato selezionato come social cache ma ogni arco nella sua ego network è coperto da una social cache o $White$ se v non è una social cache e c'è almeno un arco nella sua ego network non coperto da social cache. L'algoritmo si articola in due fasi: nella prima ogni nodo della rete seleziona il vicino con punteggio di Social Score più alto come social cache per la propria ego network. La seconda fase, del tutto simile a quella dell'algoritmo originale, viene eseguita solo dai nodi che non sono stati già eletti social cache e realizza una copertura degli archi,

identica a quella discussa nel Paragrafo 2.5, che garantisce il soddisfacimento dei vincoli di *trust*: per ogni nodo u si calcola $ratio(u)$, cioè la frazione di nodi della propria ego network E_u non coperti da social cache. Tale valore è definito come

$$ratio(u) = \frac{span(u)}{size(E_u)} \tag{2.15}$$

dove $span(u)$ è il numero di archi della propria ego network non coperti da social cache e $size(E_u)$ è il numero di archi della ego network del nodo u. Se $ratio(u)$ è maggiore di una certa soglia γ il nodo elegge se stesso come social cache. Tale fase viene ripetuta decrementando a ciascuna iterazione il valore γ di $RATIOSTEPSIZE$ finché tutti gli archi non risultano coperti. L'obiettivo di questa seconda fase è realizzare una copertura di tutti gli archi con un numero di social cache relativamente basso. Mostriamo di seguito lo pseudocodice della fase 1 dell'algoritmo in Algorithm 5 e quello della fase 2 in Algorithm 6.

Algorithm 5 SSSC - Fase 1

```
if N_u − {u} ≠ ∅ then
    get SS_v ∀v ∈ N_u
    select n | SS_n = max_{v∈N(u)}{SS_v}
    mark n as BLACK
end if
```

Algorithm 6 Social Score Algorithm - Fase 2

```
if span(u) = 0 then
    mark u as Grey;
else
    repeat
        compute ratio(u);
        if ratio(u) > γ then
            mark u as Black; (*social cache*)
            exit;
        else
            γ− = RATIOSTEPSIZE;
        end if
    until span(u) > 0
    if not Black then
        mark u as Grey;
    end if
end if
```

In definitiva l'obiettivo della prima fase è quello di selezionare un insieme di buoni punti di memorizzazione nella rete per i dati dei peer e quello della seconda di garantire una copertura del grafo che consenta il soddisfacimento dei vincoli di *trust*.

2.6 Availability in OSNs

Lo studio della availability degli utenti delle OSNs, e dei sistemi P2P in genere, sta acquisendo negli ultimi anni un'importanza sempre crescente per lo studio di molti problemi. Conoscere le caratteristiche delle sessioni degli utenti di un sistema può avere un impatto determinante sulla complessità di molti problemi, a partire da quello della diffusione delle informazioni all'interno della rete e dal problema della replicazione dei dati. Lo studio di sistemi P2P di tipo diverso ha mostrato come le caratteristiche delle sessioni degli utenti siano estremamente eterogenee e come, per progettare un algoritmo per una data OSN, sia necessario tener conto delle caratteristiche di disponibilità degli utenti del sistema stesso.

Negli ultimi anni sono stati pubblicati diversi lavori sulla availability degli utenti nei sistemi P2P. Una delle proposte maggiormente innovative di questo lavoro di tesi è quello di proporre un modello di social caching che, a differenza di tutti i lavori precedenti, tenga conto per l'elezione dei peer che dovranno assumere il ruolo di cache anche delle caratteristiche temporali dei nodi. Risulta infatti evidente come sia poco utile eleggere come social cache nodi che trascorrono online periodi brevi di tempo. In questa tesi è stato quindi condotto un approfondito studio (descritto nella prima parte del Capitolo 5) sulle caratteristiche dei periodi online e offline degli utenti di diverse OSNs, al fine di comprendere quanto il loro comportamento sia periodico e predicibile. Sarà in questo modo possibile sfruttare le caratteristiche temporali di ciascun nodo nel nostro algoritmo di elezione.

Lo studio del comportamento degli utenti nelle OSNs è un argomento di studio piuttosto recente. Nel seguito mostreremo i principali risultati ottenuti da lavori precedenti, che sono stato il nostro punto di partenza nell'analisi del Capitolo 5.

Golder *et al.* [21] sono stati tra i primi a condurre uno studio empirico sulle caratteristiche delle sessioni e delle interazioni degli utenti in una OSN. Sebbene il loro lavoro sia basato su dataset estratti da una versione di Facebook piuttosto diversa da quella attuale, in cui la social network veniva utilizzata esclusivamente da studenti di college americani, le conclusioni ottenute si sono rilevate un ottimo punto di partenza per la

prosecuzione dello studio in questo settore. Gli autori sono stati infatti tra i primi a rilevare una evidente periodicità nel numero di connessioni degli utenti, regolata fondamentalmente dai ritmi di vita nei campus. Sono stati rilevati comportamenti periodici sia su base giornaliera (picchi di connessioni in determinate ore del giorno, variazione del numero di utenti connessi in base all'alternarsi del giorno e della notte, etc.) sia su base settimanale (comportamenti diversi da quelli tipici durante i week-end, un più alto numero di connessioni e messaggi all'inizio della settimana, etc.). È stato inoltre osservato come gruppi di studenti facenti parte dello stesso college tendano ad avere comportamenti più simili tra loro rispetto a studenti di altre scuole. Queste osservazioni, se pur effettuate in un contesto piuttosto diverso rispetto alle OSNs attuali, hanno spinto numerosi ricercatori a rivalutare l'importanza del comportamento degli utenti all'interno di un sistema e sono stati il punto di partenza per studi successivi.

In [16] l'autore vuole dimostrare come le caratteristiche di disponibilità dei dati in un sistema P2P dipendano, oltre che dalle caratteristiche architetturali del sistema stesso, dalla availability degli utenti che vi partecipano. L'analisi viene condotta sulla base dello studio della disponibilità di host con caratteristiche eterogenee (host di Microsoft, Gnutella, Napster e di Internet in generale) ed ha lo scopo di individuare comportamenti ciclici e periodici degli utenti. L'autore considera la traccia dei periodi online di ogni nodo come un segnale digitale dove l'n-simo bit è pari a 1 se il nodo si trova online durante l'ora n. Applica poi una decomposizione di Fourier a ogni segnale, determinando l'insieme di onde sinusoidali con somma pari al segnale originale. Nodi che presentano picchi negli spettri delle frequenze giornaliere e settimanali hanno comportamenti periodici. L'autore individua quindi due distinti gruppi di utenti: un gruppo si trova sempre online, mentre l'altro alterna periodi online e offline, con differenti caratteristiche di periodicità.

Uno studio sull'impatto della disponibilità degli utenti nelle OSNs è presentato in [8]. L'analisi viene effettuata su una traccia raccolta da MySpace attraverso un procedimento di crawling con una frequenza di 10 minuti. Gli autori sono interessati ad analizzare le caratteristiche di availability degli utenti all'interno della rete sociale e in particolare studiano la presenza online di un utente in relazione a quella dei propri amici. Il risultato più significativo ottenuto è senz'altro l'osservazione per cui la presenza online degli utenti sembra essere correlata con quella dei propri amici: in media un utente ha il 42% di probabilità in più di trovarsi online se almeno il 50% dei suoi utenti risultano connessi. Viene inoltre osservato come la conoscenza della availability dei peer risulti di fondamentale importanza nel processo di diffusione dell'informazione all'interno della rete. Il passaggio dell'informazione a utenti che hanno trascorso online una frazione di tempo più alta garantisce infatti che l'informazione raggiunga un più alto numero di

utenti in un periodo inferiore di tempo.

Un'analisi delle caratteristiche principali di quattro diverse OSNs è presentato in [24]. Gli autori tentano di individuare caratteristiche comuni e peculiarità di MySpace, Bebo, Tagged e Netlog attraverso l'analisi di dataset distinti ottenuti attraverso un procedimento di crawling. Si rilevano ad esempio un numero di sessioni giornaliere per utente piuttosto basso (sempre inferiore a 3) mentre una durata media della sessione piuttosto diversificata tra una OSN e l'altra (si va dai circa 15 minuti di MySpace agli oltre 100 minuti di Bebo). Gli autori hanno il merito di riuscire comunque a caratterizzare il comportamento complessivo degli utenti delle diverse OSNs: si osserva infatti come i periodi di utilizzo del sistema possano essere modellati attraverso una distribuzione di Weibull e che la durata delle sessioni e il loro numero seguono una distribuzione power-law.

Sono stati proposti anche diversi sistemi che sfruttano availability-pattern in Friend-to-Friend (F2F) systems. In questi sistemi viene data un'alta importanza al livello di confidenzialità tra gli utenti. Si può dimostrare, come fatto in [22], che tali sistemi sono caratterizzati da un alto livello di correlazione tra la disponibilità di utenti tra cui esiste una relazione di amicizia. Inoltre i gruppi di amici hanno generalmente cardinalità bassa. Viene proposto uno studio sulla correlazione tra i periodi online e offline dei diversi utenti in passato, introducendo un indice di Presence Matching tra coppie di utenti (PM) e in gruppi di utenti (Group Presence Matching, GPM). Grazie a tali misure è possibile introdurre un algoritmo che riesce a stimare con grande accuratezza il grado di ridondanza necessario per la replicazione di ogni informazione, fornendo grande beneficio all'intero sistema di memorizzazione.

Anche in altri contesti sono stati proposti sistemi che tenessero conto della disponibilità dei peer della rete. In [39] viene proposto My3, una DOSN di tipo privacy-friendly. Vengono analizzate tracce relative a Facebook e Twitter e il sistema sembra riuscire a offrire un alto livello di disponibilità dei dati con un basso numero di copie del profilo di un utente. Con un tempo giornaliero medio online di circa 40 minuti per utente (valori simili a quelli misurati per Facebook) il sistema fornisce una disponibilità dei dati di oltre il 90% con una media di 4-5 repliche per profilo. Le repliche vengono inoltre affidate a utenti fidati. Tuttavia alcuni aspetti dell'approccio proposto non appaiono del tutto convincenti: in particolare l'idea di stimare i periodi online degli utenti (che non sono presenti nei dataset originali) in base ai momenti a cui avvengono le interazioni tra essi sembra essere piuttosto approssimativa.

2.6.1 Availability prediction

Gli studi sulla possibilità di prevedere la presenza futura degli utenti di un sistema P2P sulla base del loro comportamento passato sono molto meno numerosi di quelli sulla loro generica disponibilità. In [7] lo scopo degli autori è trovare peer che si comportano secondo un dato availability pattern. In particolare i nodi ricercano partner per due diverse tipologie di problemi noti come *disconnection matching* (un peer cerca un partner che si disconnetterà circa nello stesso momento) e *presence matching* (un peer cerca un partner che sarà online insieme a lui in futuro). Per entrambi questi scopi è necessario prevedere il comportamento futuro dei peer sulla base del loro comportamento passato. Gli autori propongono di usare un semplice predittore binario che predice la presenza online di un peer in una finestra temporale di 10 minuti: la predizione viene fatta sulla base del numero di giorni nella settimana passata in cui il peer era online nella stessa finestra temporale. Se il peer era online nella stessa finestra temporale in almeno 5 giorni su 7 della scorsa settimana allora si stima che il peer sarà online. Gli autori utilizzano una traccia di eDonkey per verificare l'efficacia dei predittori creati, che viene filtrata per rimuovere i peer che non sono mai stati online nell'intervallo considerato. Si ottiene quindi una traccia filtrata di circa 12 milioni di peer, la cui predicibilità generale è comunque poco elevata. Gli autori riescono però a estrarre un ridotto *predictable set* di nodi con comportamento predicibile filtrati in base a una serie di parametri. Tale insieme contiene solo 19 600 peer, ma gli autori ritengono di poter aumentare la cardinalità di tale insieme raffinando la tecnica usata per estrarre il dataset e utilizzando predittori più complessi. La conoscenza dei periodi online anche di un numero ristretto di peer permette comunque di migliorare le performance di diverse applicazioni: le prestazioni sia di un semplice algoritmo di task scheduling sia di un piccolo sistema di file-storage risultano sensibilmente incrementate.

In [37] vengono proposti dei predittori più complessi per predire la presenza online dei peer in sistemi distribuiti. Si analizza in particolare la predicibilità di nodi di PlanetLab, Microsoft Corporation e Overnet. Il comportamento dei peer di Overnet risulta sostanzialmente impredicibile, dato lo scarso tempo trascorso online. I peer di PlanetLab e Microsoft Corporation vengono classificati in diversi gruppi a seconda delle loro caratteristiche: si identifica un insieme di peer sempre online, un insieme di peer sempre offline, alcuni nodi con comportamento periodico, altri caratterizzati da sessioni lunghe e infine gruppi di peer con comportamenti disomogenei. Si osserva che i nodi di PlanetLab e quelli di Microsoft Corporation risultano partizionati nelle diverse categorie in modo assai diverso, sottolineando come le caratteristiche di disponibilità dei peer vari

in maniera consistente da un sistema all'altro. Infine i risultati di predicibilità trovati vengono applicati a tre scenari reali: si propone innanzitutto un sistema di *replica placement* che tenga conto della availability dei peer. Il posizionamento delle repliche delle informazioni su nodi con particolari caratteristiche di disponibilità permette di ridurre sensibilmente il numero di copie di un'informazione necessarie a mantenere un alto livello di availability globale. Le informazioni sulla disponibilità dei nodi può inoltre essere sfruttata sia per la diffusione delle informazioni in Delay Tolerant Network, sia per la modellazione della diffusione di un virus all'interno di una popolazione.

2.7 Temporal Networks

Abbiamo visto come la dinamicità dei peer all'interno di una rete sociale rivesta un ruolo fondamentale in numerosi problemi, che vanno dalla diffusione delle informazioni nella rete alla garanzia della disponibilità dei dati degli utenti offline. Presentiamo ora brevemente le *temporal network*, uno strumento che ci permette di caratterizzare in modo più preciso il comportamento temporale di un nodo all'interno della rete. Utilizzeremo le *temporal network*, seppur parzialmente, nel Capitolo 5 per studiare il comportamento dei peer in OSNs reali.

Una gran varietà di sistemi presenti in natura può essere rappresentata attraverso grafi di vertici connessi da archi. Il principale vantaggio derivante dall'utilizzo di grafi è la possibilità di modellare il comportamento dinamico del sistema senza dover studiare tutte le attività in esso effettivamente presenti. È infatti possibile studiare ad esempio quali sono e come si influenzano le diverse componenti di un sistema o, attraverso misure di centralità, quali nodi hanno un ruolo di particolare rilevanza all'interno di esso. Tuttavia sta emergendo negli ultimi decenni un numero sempre crescente di sistemi in cui la dimensione temporale gioca un ruolo di fondamentale rilevanza e dove siamo interessati allo studio di pattern che sono in qualche modo dipendenti dal tempo, come ad esempio la frequenza o la durata dei contatti tra i nodi del sistema. Per rispondere a tale esigenza sono stati proposti, con soluzioni lievemente diverse e con una terminologia ancora poco uniforme, quelli che buona parte degli studiosi del settore definiscono *temporal network*, dei grafi in cui si rappresentano esplicitamente i momenti in cui gli archi sono attivi. Questo strumento permette da un lato di inserire nel modello un ordinamento temporale tra le connessioni e dall'altro di introdurre nuove misure che diano una stima del ruolo che i nodi (o gruppi di nodi) hanno nel sistema tenendo conto anche del fattore tempo.

Numerosi sono i campi in cui è stato proposto con successo l'uso di reti temporali. Senza la pretesa di essere in nessun modo esaustivi, citiamo alcuni di quelli che sembrano

essere di particolare rilevanza per mostrare come le reti temporali possano diventare uno strumento di ampio utilizzo. Le *temporal network* sono state proposte per modellare varie forme di comunicazione dell'informazione, sia di tipo 1 a 1 (come lo scambio di e-mail o chiamate telefoniche) che 1 a molti (come nel broadcast). Possono essere utilizzati in modi diversi anche per rappresentare la distanza fisica a cui due *device* elettronici possono entrare in contatto (si pensi al Bluetooth). Sono state impiegate inoltre in vari ambiti scientifici che vanno dalle reti di regolazione genica alla rappresentazione di interazioni molecolari nelle reazioni chimiche. Infine possono essere un modello per estendere le reti ecologiche, in cui per esempio la relazione preda-predatore può modificarsi con lo scorrere delle stagioni.

Il resto di questo paragrafo è organizzato come segue: introdurremo inizialmente il formalismo utilizzato per rappresentare le reti temporali insieme ad alcune definizioni generali, per discutere quindi una serie di misure utilizzate per modellare la struttura topologico-temporale di un sistema dinamico. Analizzeremo infine in che modo le *temporal network* possano essere utilizzate per modellare il comportamento di reti complesse.

2.7.1 Notazione

Presentiamo le *temporal network* utilizzando la terminologia proposta in [27]. Le reti temporali che consideriamo possono essere divise in due classi corrispondenti alle due rappresentazioni proposte in Fig.2.2.

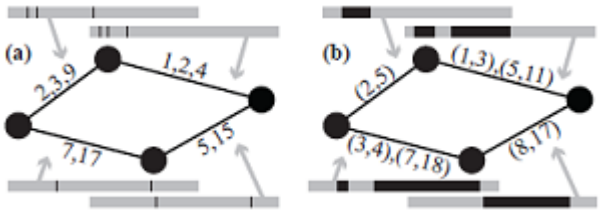


Figura 2.2: Sequenze di contatti (a) e grafo a intervalli (b)

Nella prima rappresentazione abbiamo un insieme di n vertici V che interagiscono tra loro in certi istanti di tempo; la durata dei contatti viene considerata trascurabile. In questo caso il sistema può essere rappresentato come una *sequenza di contatti*. Il grafo temporale viene definito come $G =< V, E, C >$, dove V sono i vertici (o nodi), E sono gli archi e C sono i contatti, rappresentati come triple (i, j, t) dove i e $j \in V$ e t denota il tempo a cui avviene il contatto. Nella seconda rappresentazione, detta *grafi*

a intervalli, gli archi non sono attivi in determinati istanti di tempo, ma in intervalli $T_e = \{(t_1, t'_1), ..., (t_n, t'_n)\}$ dove le parentesi indicano i periodi di attività dei contatti. Per rappresentare questa differenza i contatti vengono espressi non più tramite triple, ma attraverso quadruple $(i, j, t, \delta t)$ dove δt rappresenta la durata del contatto. Per i grafi a intervalli viene fatta l'ulteriore assunzione, in verità piuttosto restrittiva, che non ci siano intervalli sovrapposti per uno stesso nodo, si suppone cioè che ciascun vertice possa essere in ogni momento in contatto attivo con un solo altro vertice. Generalmente si considerano contatti non orientati, due contatti (x, y, t) e (y, x, t) rappresentano cioè lo stesso contatto. L'estensione a grafi orientati non presenta tuttavia particolari problemi. Infine, come per i grafi statici, è utile introdurre una funzione indice che dica se una coppia di vertici è connessa a un determinato istante di tempo:

$$a(i, j, t) = \begin{cases} 1 \text{ if } i \text{ e } j \text{ sono connessi al tempo } t \\ 0 \text{ altrimenti} \end{cases}$$

A conclusione di questa sezione mostriamo un esempio intuitivo del perché un grafo statico non riesca a rappresentare tutte le caratteristiche modellate da un grafo dinamico. Consideriamo la rete temporale rappresentata graficamente nella parte superiore di Fig.2.3 (l'immagine è stata adattata da [46]) e il grafico statico (in basso) ottenuto dall'aggregazione dei diversi snapshot temporali della rete. Appare ovvio, ad esempio, che mentre esiste un cammino dal nodo A al nodo E nel grafo statico, non esiste invece alcun cammino temporale che colleghi tali due nodi. Un grafo aggregato non è quindi in grado di modellare vari aspetti di una rete dinamica di importanza rilevante, come ad esempio la frequenza e la durata dei contatti. Ciò mette in evidenza come sia necessario introdurre delle nuove definizioni per i comuni concetti di cammino, cammino minimo, componenti connesse, indici di centralità, etc., che considerino il tempo come ulteriore dimensione delle reti temporali.

2.7.2 Misure per la struttura topologico-temporale

La struttura topologica di una rete statica può essere caratterizzata da un ampio numero di misure basate sulle connessioni tra nodi o tra insiemi di nodi. Come visto al termine della sezione precedente, quando aggiungiamo la dimensione temporale alla rete, molte di tali misure devono essere rivalutate o riviste. L'esempio più banale di ciò è il fatto che qualsiasi cammino sulla rete deve seguire una sequenza di contatti ordinati temporalmente. Per semplicità di notazione nel seguito considereremo reti rappresentate

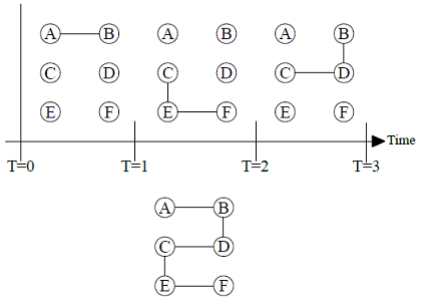


Figura 2.3: Un grafo temporale e il corrispondente grafo statico aggregato

attraverso *sequenze di contatti*, ma le definizioni date possono essere estese in modo del tutto intuitivo al caso dei *grafi a intervalli*.

Time-respecting paths

Definiamo *time-respecting path* una sequenza di contatti con tempi di contatto non decrescenti. Nell'esempio di Fig.2.3 (A, B, 0), (B, D, 2) è un *time-respecting path* mentre (B, C, 2), (C, E, 1) non lo è. La differenza principale tra i cammini nelle reti statiche e i *time-respecting path* è il fatto che questi ultimi non sono transitivi: l'esistenza di *time-respecting path* tra i e j e tra j e k non assicura l'esistenza di un *time-respecting path* tra i e k.

I *time-respecting path* definiscono quindi quali vertici possono essere raggiunti dagli altri nodi all'interno di una determinata finestra di osservazione $t \in [t_0, T]$. In aggiunta è possibile aggiungere dei vincoli sul tempo che i cammini possono trascorrere nei vertici, ovvero sui tempi tra due contatti consecutivi in un cammino. Questo ad esempio può essere utile per studiare il verificarsi di interazioni tipiche tra nodi. Consideriamo ad esempio uno studio sulle chiamate fatte tra gli utenti di una compagnia telefonica. Vogliamo sapere quanto frequentemente a una chiamata di un utente x a un utente y segue una chiamata "di risposta" dell'utente y all'utente x. È ovvio che se la chiamata avviene oltre una certa soglia temporale essa non è più da considerarsi come "di risposta" alla precedente, ma come un ulteriore successivo contatto tra gli utenti in questione.

Distanze, latenze e cammini minimi

Per le reti statiche si definisce la *geodesic distance* tra due vertici come la lunghezza del cammino minimo che li collega. Per le reti temporali è possibile definire il concetto di

cammino minimo sia in termini di numero di hop (o di archi di peso minimo) in modo analogo al caso statico, sia in termini di lunghezza temporale del cammino, definita come la differenza temporale tra l'ultimo e il primo contatto del cammino. Il tempo minimo in cui un nodo i può raggiungere un nodo j è detta la loro *latenza* (o *distanza temporale*). Tuttavia anche la latenza è dipendente dal tempo e in particolare essa può variare in base al tempo in cui si inizia a percorrere il cammino. Consideriamo quindi un nodo i al tempo t in una rete temporale su cui si diffondono informazioni. Denotiamo con $\phi_{i,t}(j)$ l'ultimo istante di tempo precedente a t in cui l'informazione può essere partita da j per raggiungere i entro il tempo t. Chiamiamo tale misura la *vista* di i dell'informazione di j al tempo t e chiamiamo *latenza dell'informazione* di j rispetto a i al tempo t il valore $\lambda_{i,t}(j) = t - \phi_{i,t}(j)$. Questa esprime quanto vecchia è la misura di i proveniente da j al tempo t. Un'altra misura a cui siamo interessati, detta *latenza media*, è una media del tempo impiegato da un nodo i per raggiungere un nodo j, che può essere calcolata come una media di tutti i *time-respecting path* minimi tra due nodi in funzione del tempo. Tuttavia una tale misura può essere difficile da calcolare perché verso la fine della finestra di osservazione la latenza diviene infinita, dato che i cammini non hanno tempo sufficiente per essere completati entro il termine della finestra temporale. In ogni modo si definisce *latenza caratteristica* la distanza temporale (latenza) media tra ogni coppia di nodi del grafo, ovvero:

$$L = \frac{1}{N(N-1)} \sum_{ij} d_{ij}$$

dove con d_{ij} indichiamo le distanze temporali.

Temporal motifs

Introduciamo brevemente anche il framework dei *temporal motif* per studiare la struttura topologico-temporale di reti dinamiche in cui gli eventi dei singoli nodi non si sovrappongono nel tempo. I *temporal motif*, presentati da Kovanen et al. in [29], sono classi di sequenze di eventi simili, dove la similarità si riferisce non solo alla topologia ma anche all'ordine temporale degli eventi.

Molte reti di grandi dimensioni presentano proprietà simili su scala globale, come distribuzioni caratteristiche dei gradi dei nodi, lunghezza ridotta dei cammini tra coppie di nodi o alto numero di triangoli. A livello mesoscopico tali reti presentano una maggiore varietà ma presentano comunque *motif* particolarmente significativi, ovvero piccoli sottografi topologicamente equivalenti che possono rappresentare interazioni sociali di

particolare rilevanza. Formalmente i *motif* sono definiti come classi di grafi isomorfi in cui i nodi sono tutti temporalmente connessi. Due eventi sono detti $\Delta t-$*connessi* se esiste una sequenza di eventi $e_i = e_{k_0} e_{k_1} ... e_{k_n} = e_j$ tale che tutte le coppie di eventi consecutivi sono $\Delta t-$*adiacenti*, dove due eventi vengono detti $\Delta t-$*adiacenti* se hanno almeno un nodo in comune e la differenza di tempo tra la fine del primo evento e l'inizio del secondo è al più Δt. In [29] gli autori propongono un algoritmo per il riconoscimento di *temporal motif* a partire da sottografi massimali connessi.

Cerchiamo di spiegare meglio con un esempio cosa siano questi *motif*. Utilizziamo un dataset contenente le chiamate di un singolo operatore di telefonia mobile in un periodo di 120 giorni. Vogliamo scoprire i motivi ricorrenti che coinvolgono 3 eventi consecutivi, considerando una finestra temporale di $\Delta t = 10min$. I motivi più ricorrenti sono, come mostrato in Fig.2.4 (adattata da [29]), quelli più semplici, che coinvolgono nella conversazione solo due utenti, mentre quelli meno comuni sono *motif* triangolari. Tramite i *temporal motif* è quindi possibile studiare interazioni sociali tipiche di un certo sistema, studiando in questo modo il comportamento caratteristico degli utenti che ne fanno parte.

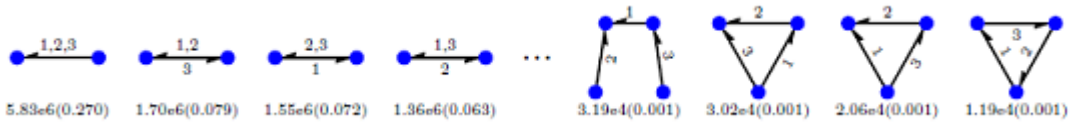


Figura 2.4: I quattro *motif* formati da 3 eventi più comuni (a sinistra) e meno comuni (a destra) riscontrati nei dati

2.7.3 *Temporal Networks* e reti complesse

In questo paragrafo abbiamo presentato le *temporal network*, mostrando come il loro utilizzo permetta di modellare caratteristiche del sistema che non potevano essere rappresentate con le sole reti statiche. Ancora poco è tuttavia presente in letteratura su come le reti temporali possano essere concretamente implementate. In particolare sembra ancora aperto, o studiato solo in parte, il problema di utilizzare le reti temporali per modellare il comportamento di una DOSN. Un interessante campo di ricerca potrebbe essere lo studio di come l'uso di tali reti possa migliorare le performance di tutti gli algoritmi che risentono negativamente del passaggio offline di alcuni dei nodi. È tuttavia importante notare come molte delle misure introdotte con le *temporal network* si

basino sulla conoscenza dell'intera rete e che quindi sono necessari algoritmi distribuiti per approssimare tali misure.

Le reti temporali, che sono ancora oggetto di uno studio intensivo da parte di diversi gruppi di ricerca, si stanno comunque rivelando uno strumento di fondamentale importanza per modellare il crescente numero di sistemi dinamici in cui il tempo gioca un ruolo fondamentale nel riconoscimento di pattern significativi. Le metriche a esse correlate costituiscono una valida alternativa ai metodi standard per lo studio di sistemi che evolvono nel tempo e potranno certamente produrre un contributo fondamentale alla nostra comprensione dei sistemi complessi.

Capitolo 3

Data persistence in DOSNs dinamiche: una proposta

Nel Paragrafo 2.5 abbiamo discusso come l'utilizzo di social cache possa ridurre il costo della diffusione delle informazioni in una OSN. In questo capitolo mostreremo come sia possibile utilizzare una tecnica simile anche per il problema della persistenza dei dati. Dopo aver presentato il modello di social overlay utilizzato e aver discusso sull'opportunità di rappresentare le ego network dei nodi attraverso grafi orientati o non orientati (Paragrafo 3.1), mostreremo nel Paragrafo 3.2 come sia possibile adattare l'idea base del social caching per garantire la persistenza dei dati all'interno della rete. Presenteremo quindi il modello che assumiamo per lo studio del problema e enunceremo quali sono gli obiettivi che ci poniamo di raggiungere (Paragrafo 3.3) per concentrarci poi sulle caratteristiche che i nodi devono avere per rappresentare buoni punti di memorizzazione nella rete. Illustreremo quindi i risultati ottenuti nello studio della dinamicità degli utenti di alcune OSNs: tali risultati sono stati utilizzati per formulare un algoritmo di elezione dei punti di memorizzazione dei dati dei nodi che considera la dinamicità dei peer all'interno della rete (Paragrafo 3.6). Nel Paragrafo 3.7 vengono infine presentati gli algoritmi necessari a gestire la dinamicità delle social cache: attraverso una copertura del grafo sociale garantiremo che i dati degli utenti offline siano comunque accessibili agli utenti della rete che ne hanno diritto. È importante notare che la dinamicità delle social cache è un fattore che veniva quasi completamente ignorato in tutte le precedenti proposte.

3.1 Modello per la Social Overlay

Per descrivere i servizi offerti da una DOSN è necessario innanzitutto definire l'*overlay network*, ovvero la rete di contatti presenti all'interno del sistema. Molte OSNs definiscono un numero massimo di relazioni di amicizia che un contatto può instaurare, tale valore è solitamente molto alto (5000 amici per Facebook), spesso di gran lunga superiore al numero massimo di connessioni che un dispositivo è in grado di gestire contemporaneamente. Tuttavia non tutte le relazioni di amicizia hanno le stesse caratteristiche. Come osservato da Dunbar et al. in [42] esiste un limite cognitivo al numero di persone con cui un individuo può mantenere relazioni sociali stabili: è pertanto possibile tenere traccia dei soli individui con cui un nodo possiede una relazione sociale attiva. In [17] viene introdotto il *Dunbar's number*, definito come il numero di persone che un individuo conosce e con cui mantiene contatti sociali stabili. Tale numero risulta pari a 150.

Definiamo pertanto la nostra rete sociale globale attraverso un grafo $G = (V, E)$ dove V sono gli utenti della rete e E i legami sociali tra di essi. Per descrivere le caratteristiche dei legami sociali di un individuo andiamo ad aggiungere a tale modello un'ulteriore struttura, detta *ego network*, che descrive la rete sociale formata dall'individuo (ego) e dalle persone con cui esso è in contatto (alter) [33]: essa contiene sia le relazioni tra l'ego e gli alter che quelle esistenti tra gli alter. Queste reti sono state attentamente studiate dal punto di vista antropologico e presentano delle caratteristiche peculiari. Innanzitutto la proprietà fondamentale di ogni interazione tra due individui è il *tie-strength* che è una misura della forza del legame esistente tra due individui. Il valore quantitativo da attribuire a tale misura può dipendere da una molteplicità di fattori tra cui la durata dei contatti, la loro frequenza e il tipo di relazione.

Gli studi antropologici condotti su relazioni sociali offline hanno portato alla definizione di un modello di ego network che comprende diversi anelli concentrici e organizzati gerarchicamente su cui si dispongono gli alter in base al valore del *tie-strength*. In [45] Dunbar et al. identificano quattro livelli denominati, dall'interno verso l'esterno, *support clique, sympathy group, affinity group* e *active network* e caratterizzati da un livello di confidenza decrescente.

Il *support clique* è il livello più interno della struttura concentrica, formato dagli individui a cui l'ego chiederebbe consiglio, supporto e aiuto in caso di una grave emergenza emozionale o finanziaria ed è composto in media da 5 membri. Si stima che l'ego abbia contatti con gli individui facenti parte di questo gruppo almeno una volta la settimana. Il livello immediatamente più esterno è quello del *sympathy group*, definito come l'insieme di individui con cui l'ego ha contatti almeno mensilmente. Tale gruppo è formato in

media da 15 individui. Muovendoci ancora verso l'esterno troviamo l'*affinity group*, che è caratterizzato in maniera meno precisa dal punto di vista della frequenza dei contatti. Si stima comunque che abbia una dimensione media di circa 50 persone. Infine tutti gli individui con cui l'ego ha una relazione stabile e che non rientrano nei precedenti livelli fanno parte dell'*active network*.

Si osserva che esiste un rapporto pressoché costante tra la dimensione di un livello e quella del livello immediatamente più interno: tale rapporto è pari a circa 3. La struttura descritta è mostrata in Figura 3.1.

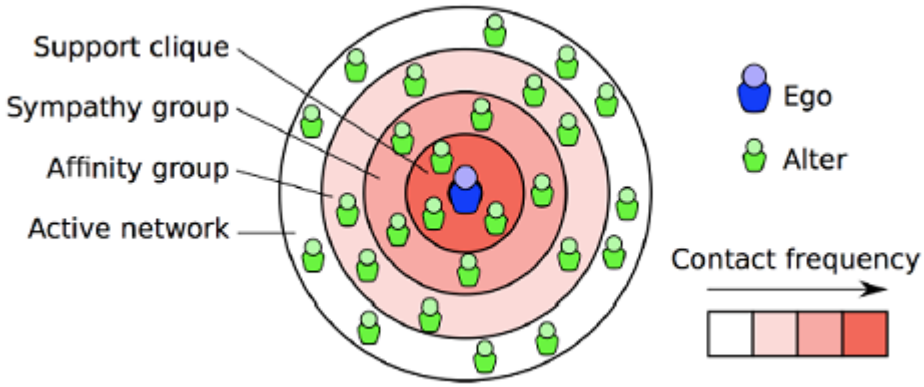


Figura 3.1: Cerchia di Dunbar

Uno dei principali vantaggi di tale rappresentazione è il fatto che le relazioni in essa presenti sono caratterizzate da un certo livello di fiducia, dato che le cerchia di Dunbar più vicine all'ego sono caratterizzate da un grado di confidenza sempre più alto. Si è osservato inoltre che, sia in ambienti virtuali che in ambienti reali, le relazioni sociali più stabili sono instaurate con utenti che presentano caratteristiche di similarità molto forti con l'ego: si parla perciò di *omofilia* tra individui di una ego network. Si può osservare come l'alto grado di similarità tra l'ego e gli individui dei livelli più interni della ego network riguardi non solo l'interesse comune verso determinati argomenti ma anche comportamenti simili all'interno della rete sociale.

Sono stati effettuati numerosi studi che dimostrano la presenza di caratteristiche comuni nelle ego network online e in quelle offline [3]. Ciò che differenzia le ego network del primo e del secondo tipo è naturalmente la definizione del *tie-strength*. Nelle OSNs, come accennato in precedenza, è possibile estrarre un valore numerico che dia una stima di tale misura sulla base di un certo numero di fattori. Nel modello che implementeremo, come in molti lavori sull'argomento (si veda ad esempio [30]), utilizzeremo come misura

della forza di legame la frequenza di contatto, anche se alcuni studi (come ad esempio [34]) hanno dimostrato che in molte reti sociali offline tale misura sovrastima la forza del legame tra vicini di casa o colleghi.

Vediamo quindi come sia possibile modellare una ego network Dunbar-based. Una prima rappresentazione che è stata proposta è costituita da un grafo diretto e pesato $\vec{G} = (V, \vec{E})$, dove V è l'insieme degli utenti e $\vec{E}$ è l'insieme di interazioni tra di essi. Ad ogni link è associato un peso corrispondente al valore del *tie-strength* e che quantifica il grado di interazione tra i due utenti. Un secondo modello per una ego network Dunbar-based è costituito da un grafo non orientato $G = (V, E)$ dove V è l'insieme degli utenti e E sono le relazioni sociali definite all'interno delle cerchia di Dunbar. Anche in questo caso a ogni arco non orientato è associato un peso p_{ij} che può essere calcolato con la media dei pesi p_{ij} e p_{ji} delle corrispondenti interazioni in $\vec{E}$. Per ogni vertice $u \in V$ è definita la corrispondente ego network $EN(u) = (V_u, E_u)$ dove l'insieme V_u è definito come $V_u = \{v \in V | (u, v) \in E\}$ e $E_u = \{(a, b) \in E | a = u \vee b = u \vee \{a, b\} \subseteq V(u)\}$. L'ego network di un utente è quindi una rete che ha come nodi l'utente stesso e tutti gli utenti della rete con cui ha contatti stabili. Tra ogni coppia di nodi di una ego network c'è un arco se esiste una relazione sociale stabile tra essi: una rappresentazione grafica di una ego network è data in Figura 3.2

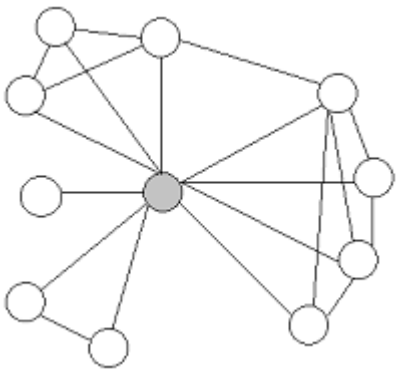


Figura 3.2: Possibile ego network per il nodo in grigio

3.1.1 Uso di grafi orientati o non orientati per la rappresentazione di ego network

Nel paragrafo precedente abbiamo proposto due modelli alternativi per la rappresentazione delle ego network, il primo basato sull'utilizzo di grafi orientati e il secondo

sull'utilizzo di grafi non orientati. La scelta su quale modello utilizzare costituisce un punto chiave per l'intero lavoro e appare quindi importante discutere quale dei due modelli risulti più adatto ai nostri scopi.

Un primo paper sull'argomento presente in letteratura è [12]: in questo lavoro viene proposto un algoritmo per generare una rete sociale Dunbar-based in cui ogni nodo è connesso unicamente agli alter della propria ego network. La strategia utilizzata per l'estrazione della rete, basata alternativamente sulla ricerca di triangoli o di link che connettono nodi facenti parte di comunità diverse all'interno del grafo sociale, prevede l'utilizzo di archi non diretti. Per questo motivo la relazione "essere nella ego network" risulta simmetrica e due nodi connessi da un link si trovano nella stessa cerchia di Dunbar l'uno rispetto all'altro. L'articolo [2], realizzato dallo stesso gruppo di ricerca del precedente, introduce il concetto di *Interaction Graphs*, che non descrivono le ego network, ma contengono un link pesato per ogni arco del grafo sociale. Il peso di ciascun arco è pari al numero di interazioni avvenute tra due utenti, senza considerare il verso in cui queste avvengono. [3] propone una tecnica per derivare il valore del *tie-strength*, cioè la forza del legame tra due utenti, sulla base di un insieme di variabili: numero di like, di post, di commenti, di messaggi privati, etc. È interessante notare come ciascuno dei precedenti valori sia considerato dall'ego verso l'alter e viceversa. Questo corrisponde al fatto che stiamo descrivendo una relazione tra due individui, che è un concetto intrinsecamente bidirezionale. Tuttavia gli autori dell'articolo osservano che il numero di comunicazioni di un nodo in input e in output risulta considerevolmente diverso e che quindi esse non devono avere la stessa importanza nella determinazione del valore di *tie-strength*. La soluzione proposta non consiste tuttavia nell'utilizzare grafi orientati, ma semplicemente nel dare peso diverso alle interazioni in input e in output e di derivare così da tutte le variabili relazionali un unico peso da associare all'interazione. Infine un'interessante osservazione a sostegno dell'utilizzo di grafi non orientati è contenuta in [48]. L'articolo introduce il concetto di *Interaction Graph* dopo aver riportato un'analisi approfondita del grafo sociale di Facebook. Anche in questo caso l'*Interaction Graph* introdotto risulta non orientato. Per dimostrare che la scelta di usare un grafo non orientato è accettabile, gli autori mostrano che molto spesso c'è "reciprocità" nelle interazioni, ovvero che se un utente A scrive sul wall di B, molto spesso B risponde con un'interazione in senso inverso.

Tuttavia non mancano in letteratura articoli che seguono l'approccio opposto, cioè quello basato su grafi orientati. L'approccio utilizzato da tali lavori scaturisce in parte dalla mancanza di una definizione univoca di *tie-strength*: la definizione storica data da

Ganovetter in [19][1] mette infatti in luce come l'aspetto della reciprocità sia fondamentale, ma definizioni diverse date in un secondo tempo non fanno altrettanto. In [2], come abbiamo accennato, gli autori approssimano il *tie-strength* attraverso le interazioni tra gli utenti, senza considerare il verso in cui esse avvengono. Tuttavia la forza di un legame può essere percepita in modo diverso dai due nodi tra cui avviene l'interazione: potrebbe accadere che per il nodo A, che ha molti contatti, il peso dell'arco (A,B) sia inferiore a quello di molte altre relazioni, mentre per B, che ha molti meno amici, lo stesso valore costituisca il peso massimo nella sua intera ego network. Per questo motivo gli autori propongono di normalizzare la frequenza di contatto di ogni nodo dividendola per la frequenza massima. Segnaliamo infine l'interessante articolo [30] in cui gli autori si propongono di modellare il *tie-strength* come un indicatore delle risorse cognitive che un utente impiega in una relazione sociale. Sotto quest'ottica le comunicazioni in uscita, che sono in numero molto inferiore a quelle in entrata, devono avere un peso decisamente maggiore, perché richiedono un numero di risorse cognitive maggiore da parte dell'utente. Il *tie-strength* tra due nodi è quindi definito come

$$TieStrength_{jk} = f_{jk} + \frac{f_{jk} \cdot f_{kj}}{f_{jk} + f_{kj}} \tag{3.1}$$

dove f_{xy} rappresenta la frequenza delle comunicazioni in uscita dall'ego x e in ingresso all'ego y. Il primo addendo rappresenta quindi le comunicazioni in uscita, il secondo è massimo quando per ogni relazione esiste anche la comunicazione inversa. In questo caso quindi il grafo della rete è orientato, anche se si è voluto comunque rappresentare un certo grado di reciprocità nelle interazioni.

Abbiamo quindi visto come in letteratura per rappresentare le ego network dei nodi vengano utilizzati sia grafi orientati sia, in misura largamente predominante, grafi non orientati. Dato che lo scopo di questa tesi è la proposta di un modello originale per la persistenza dei dati in una DOSN, in cui il *tie-strength* rappresenta il livello di *trust* tra due individui, sembra ragionevole utilizzare un modello che tenga conto in qualche modo della reciprocità della fiducia tra due nodi. Queste osservazioni ci hanno fatto propendere per l'utilizzo di grafi non orientati per la rappresentazione delle ego network dei nodi.

[1] "The strength of a tie is a (probably linear) combination of the amount of time, the emotional intensity, the intimacy (mutual confiding), and the reciprocal services which characterize the tie"

3.1.2 Definizione dell'overlay P2P

A partire dall'insieme delle ego network Dunbar-based è possibile definire un'overlay P2P che rispecchi i vincoli sociali esistenti tra gli utenti. Poiché adottiamo un modello *a grafi non orientati*, e quindi la relazione "essere nella ego network" è simmetrica, l'overlay P2P contiene un link tra due peer A e B se e solo se l'utente corrispondente al peer A ha nella propria ego network Dunbar-based B e viceversa. Il numero di collegamenti che un peer deve mantenere è quindi limitato a 150. Tutti i contatti che non appartengono alla ego network Dunbar-based di un peer vengono comunque memorizzati dal peer stesso e un collegamento *on-demand* può essere stabilito con essi in caso di interazione.

Infine l'approccio è dinamico, nel senso che un contatto non appartenente alla ego network Dunbar-based può, in seguito alla presenza di un numero elevato di contatti, entrare a farvi parte e viceversa.

3.2 Il problema della persistenza dei dati

Nel Paragrafo 2.5 abbiamo introdotto una strategia per ottimizzare il numero di connessioni tra nodi in una DOSN basato sull'uso di social cache e abbiamo mostrato come l'utilizzo di un numero contenuto di social cache possa ridurre sensibilmente il numero di messaggi necessari per la diffusione di social update all'interno della rete. Secondo questo approccio le social cache vengono quindi sostanzialmente utilizzate per facilitare la diffusione dell'informazione all'interno della rete. Abbiamo però accennato al fatto che una strategia simile a quella proposta può essere utilizzata anche per un altro scopo, quello della persistenza dell'informazione in una DOSN. In un contesto dinamico infatti i dati di un utente sono disponibili, se non sono stati precedentemente "dati in consegna" ad un altro utente, solo se esso si trova online nel sistema.

Nel seguito i termini nodo, peer e utente verranno utilizzati, se non specificato diversamente, come sinonimi.

Nel modello descritto una social cache è un potenziale punto di memorizzazione per i dati dei suoi vicini offline: una social cache memorizza o è in grado di accedere *on-demand* alle informazioni dei peer a essa collegati e le rende disponibili agli utenti che ne hanno diritto quando il proprietario dell'informazione non è online nel sistema. Ciò garantisce ad esempio che i post contenuti nella bacheca di un utente siano accessibili anche quando esso si trova offline.

Il nostro approccio è basato inoltre sul concetto di *trust* tra peer, differenziandosi profondamente dalle precedenti proposte presenti in letteratura. Per noi infatti la

presenza di un nodo A nella ego network di un nodo B indica una relazione di fiducia reciproca tra A e B e il potenziale interesse di ciascun nodo verso i dati contenuti nella bacheca dell'altro. I vincoli di *trust* che vogliamo siano rispettati in ogni momento sono i seguenti:

- ogni peer è disposto a dare in consegna i propri dati solo a nodi della propria Dunbar-based ego network;
- ogni peer online vuole poter accedere, se esso stesso non li memorizza, ai dati di ogni vicino offline tramite un vicino comune che ha con esso.

Esempio 3.2.1. Consideriamo il grafo di Figura 3.3.

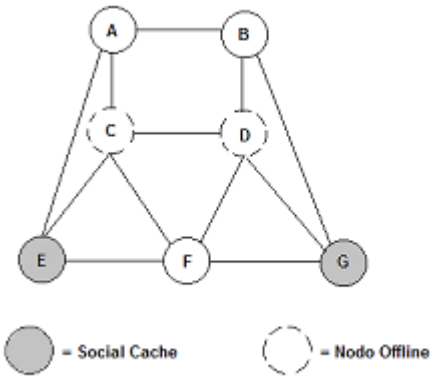


Figura 3.3: Esempio di social caching

L'obiettivo del social caching è garantire che i dati di C e D, attualmente offline, siano accessibili ai loro amici online. Supponiamo che E e G siano stati eletti social cache. E possiede quindi una copia dei dati di C e G una copia dei dati di D. In questo modo A e F possono accedere ai dati di C tramite E e B e F possono accedere ai dati di D tramite G. Ogni nodo ha inoltre dato in consegna i propri dati a un nodo della propria ego network e quindi i vincoli di *trust* sono rispettati. □

Gli obiettivi del social caching per il problema della persistenza dei dati sono quindi da un lato quello di selezionare un insieme di buoni punti di memorizzazione per i dati degli utenti e dall'altro di consentire a ciascun peer di accedere ai dati dei vicini, anche quando essi si trovano offline, rispettando un insieme di vincoli di *trust*. Il secondo dei due obiettivi viene realizzato attraverso un concetto di copertura di grafo, ispirato

a quello proposto in [26] e discusso nel Paragrafo 2.5. In questo modo si raggiunge un duplice risultato: da un lato, grazie alla struttura distribuita del nostro modello, i dati non vengono affidati a nessuna entità centralizzata, che potrebbe utilizzarli per ragioni commerciali o comunque per altri scopi non desiderati dall'utente e, dall'altro, consegnando i dati a nodi verso cui esiste una relazione di fiducia, è possibile garantire un certo grado di riservatezza senza dover contare su un meccanismo di cifratura dei dati stessi.

Infine sottolineiamo che la nostra proposta non andrà ad esaminare nel dettaglio la strategia di memorizzazione dei dati all'interno del sistema (che può comportare partizionamento e replicazione dei dati), ma garantirà più semplicemente in ogni momento l'individuazione di un insieme di nodi che garantiscano la persistenza dell'informazione nell'intera rete e la possibilità di accedervi da parte di tutti gli utenti che ne hanno diritto senza violare i vincoli di *trust*.

Tra i problemi direttamente collegati all'approccio sopra descritto ci sono i seguenti:

- definire una strategia di selezione delle social cache che si adatti all'ambiente distribuito e decentralizzato definito e tenga conto delle caratteristiche di disponibilità dei nodi all'interno della rete;
- prendere in considerazione l'aspetto della dinamicità delle social cache, introducendo un *meccanismo di rielezione* che garantisca il soddisfacimento dei vincoli di *trust* in ogni momento;
- studiare una strategia intelligente di memorizzazione dei dati e di ridistribuzione delle informazioni possedute da una cache al momento di passare offline;
- garantire la consistenza delle informazioni sociali dei nodi e delle diverse copie dei dati presenti nella rete.

Il nostro studio si è concentrato sul primo e sul secondo dei punti precedenti, analizzando in particolare come la disponibilità dei peer abbia impatto sull'elezione di buoni punti di memorizzazione per la rete. In particolare vorremmo definire una strategia distribuita per l'elezione delle social cache che tenga conto delle caratteristiche strutturali e temporali di un nodo e della dinamicità complessiva del sistema.

3.3 Modello di riferimento

Presentiamo ora il modello del sistema da noi proposto per garantire la persistenza dell'informazione in una DOSN. I nostri obiettivi sono:

1. garantire che in ogni momento i dati dei peer offline siano disponibili all'interno della rete;

2. determinare una strategia per selezionare dei punti di memorizzazione globali nella rete;

3. garantire che in ogni momento i dati di un peer p, online o offline, siano memorizzati in almeno un nodo della sua ego network, scelto sulla base delle sue caratteristiche strutturali e temporali. Tale nodo sarà un *punto di memorizzazione locale* dei dati di p;

4. garantire che in ogni momento i dati di un peer p possano essere acceduti dagli utenti che ne hanno diritto (cioè i vicini di p).

I punti 3 e 4 garantiscono il rispetto dei nostri vincoli di *trust*, enunciati nel Paragrafo 3.2.

Diamo ora una definizione formale di cosa intendiamo per *social cache* e per *punto di memorizzazione locale*.

- una *social cache* è un *potenziale* punto di memorizzazione della rete, che memorizza o è in grado di accedere ai dati dei propri vicini offline;

- il *punto di memorizzazione locale (PML)* di un nodo n è la social cache scelta da n, sulla base delle sue caratteristiche strutturali e temporali, come punto di memorizzazione dei propri dati. I dati di ogni nodo, online o offline, sono memorizzati nel proprio punto di memorizzazione locale.

Come mostrato nell'Esempio 3.3.1, una social cache può non memorizzare i dati di alcun nodo.

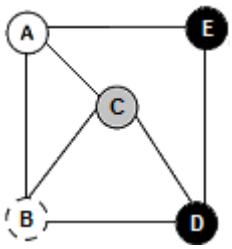


Figura 3.4: Esempio di social cache che non memorizza dati

Esempio 3.3.1. Consideriamo il grafo di Figura 3.4 in cui inizialmente i nodi sono tutti online. I nodi neri rappresentano social cache che sono anche PML per i dati di qualche nodo; supponiamo ad esempio che sia $PML_A = PML_D = E, PML_B = PML_C = PML_E = D$. Supponiamo che B decida di passare offline (nodo tratteggiato). Prima di passare offline il nodo deve garantire che i propri dati siano accessibili ai propri vicini online (cioè A, C e D) anche dopo la sua disconnessione. Dato che D memorizza i dati di B, C e D stesso potranno accedere a essi. A invece non è amico di D e ha quindi bisogno di un altro punto di accesso ai dati, che deve essere un amico comune tra A e B. L'unico amico comune è C che diventerà quindi una social cache (nodo grigio). Tuttavia non è necessario che C memorizzi i dati di B, visto che essi sono memorizzati in D, a cui C può accedere. C in questo caso ha il solo compito di fare da *social bridge* tra A e i dati di B, senza tuttavia memorizzarli effettivamente. Supponiamo invece per un momento che l'arco (C,D) non esista: in questo caso C, che verrà comunque eletto social cache, non può accedere ai dati di B tramite D quindi, al momento del suo passaggio offline, B dovrà dare una replica dei propri dati anche a C. Tuttavia in scenari reali i casi in cui è necessario replicare i dati di un nodo n che sta passando offline sono estremamente rari: i nodi hanno un numero molto elevato di amici e basta che un nodo interessato ai dati di n abbia un vicino online che può accedervi (ad esempio perché è vicino del PML di n) perché esso possa fare da *social bridge* verso i dati di n senza che essi vengano replicati. □

Definizione di copertura

Il soddisfacimento dei vincoli di *trust* viene assicurato attraverso un nuovo concetto di copertura del grafo sociale. L'algoritmo del Paragrafo 2.5.3 definiva la copertura solo per archi tra coppie di nodi online. La nostra proposta invece, che considera il dinamismo delle social cache, deve dare una definizione di copertura che tenga conto dello stato dei nodi.

Consideriamo un arco non orientato (A,B) tra due nodi nel grafo sociale; il significato di tale arco nel nostro modello è il seguente:

- A è interessato ai dati di B e vuole poter accedervi anche quando B è offline (e viceversa);
- A ha una relazione di fiducia verso B e quindi è disposto a dare ad esso in consegna i propri dati (e viceversa).

Dato un arco (A,B) esso è coperto se ciascun nodo n corrispondente a uno dei vertici dell'arco, se online, può accedere ai contenuti pubblicati dal nodo corrispondente all'altro vertice.

Possono verificarsi quattro situazioni:

- A e B sono entrambi online: in questo caso l'arco è coperto perché A può accedere direttamente ai dati di B e viceversa. Ciascun nodo può richiedere i dati all'altro nodo mediante la connessione diretta presente nel Dunbar-based overlay;
- A e B sono entrambi offline: anche in questo caso l'arco è ovviamente coperto;
- A è online e B offline: in questo caso l'arco è coperto se A è eletto social cache (in questo caso A memorizza o può accedere ai dati di B) o esiste un amico comune online che è social cache. In questo caso se A vuole accedere ai dati di B può chiederli in ogni momento a C, che è un amico comune tra A e B.
- A è offline e B online: questo caso è perfettamente simmetrico al precedente.

La Figura 3.5 rappresenta in modo sintetico la nostra nozione di copertura.

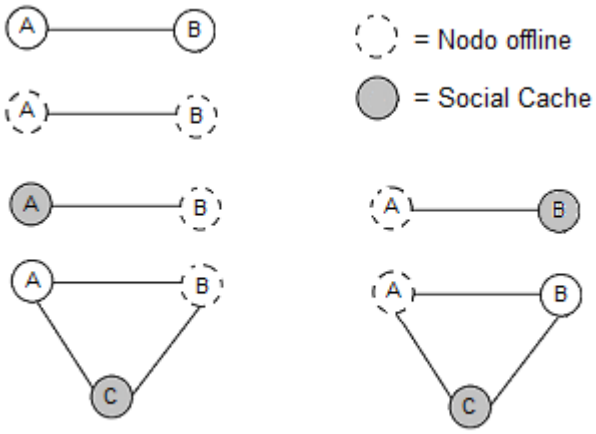


Figura 3.5: Copertura arco (A,B)

Osserviamo che *l'elezione a social cache di un nodo garantisce la copertura di tutti gli archi offline della sua ego network.*

Vogliamo inoltre che, in ogni momento, i dati di un nodo, online o offline, siano memorizzati in almeno un punto di memorizzazione: ciò garantisce un certo grado di

persistenza dell'informazione nel sistema anche in caso di fallimento inaspettato di un nodo. Inoltre tutti gli archi verso tutti i vicini offline di un nodo online devono risultare coperti, in modo che quel nodo possa accedere a tutti i loro dati.

Riepiloghiamo ora la visione complessiva del nostro modello.

Nel loro complesso le social cache costituiscono una sorta di *overlay dinamico* che garantisce la persistenza dell'informazione all'interno della rete garantendo il soddisfacimento dei vincoli di *trust*. Le social cache nel nostro modello costituiscono dei potenziali punti di memorizzazione, ma non ci siamo occupati del problema della effettiva distribuzione dei dati tra di esse. Proporremo più semplicemente una strategia per individuare buoni punti di memorizzazione per i dati dei peer della rete in un dato momento, e garantiremo che, se ciascuna social cache memorizza o è in grado di recuperare *on-demand* i dati di ciascun vicino, ogni nodo è in grado di accedere ai dati di tutti i nodi della propria ego network, anche quando essi si trovano offline. Le condizioni sopra e in particolare la copertura non possono essere ottenute con un algoritmo statico come quello proposto nel Paragrafo 2.5.3, ma devono essere ottenute con meccanismi dinamici che proponiamo in questa tesi e che verranno descritti nei Paragrafi 3.6 e 3.7.

3.4 Strategie per la selezione di social cache

Vogliamo ora indagare quali debbano essere le caratteristiche che deve avere un nodo per essere scelto come social cache a un certo istante di tempo t. Osserviamo che stiamo cercando dei punti di memorizzazione globali nella rete: una social cache infatti deve rendere disponibili ai propri vicini i dati di tutti i nodi offline nella sua ego network.

I "buoni" punti di memorizzazione vengono scelti nell'algoritmo presentato nel Paragrafo 2.5.3 sulla base del punteggio di Social Score, che combina due termini: l'*Ego Betweenness Centrality* ci fornisce una misura della centralità del nodo all'interno della rete e la $NodeAvailability$ approssima la disponibilità del peer nel sistema. Si potrebbero inoltre considerare caratteristiche hardware dei nodi, come la larghezza di banda o l'ampiezza della memoria a disposizione.

La nostra idea è che il punteggio di Social Score di un nodo debba tener conto in prima approssimazione di almeno due fattori:

- un termine che stimi la centralità di un peer nella rete: questo garantisce che tale nodo, se eletto social cache, renda disponibili i dati in esso memorizzati (o a cui può accedere) ad un alto numero di vicini garantendo il rispetto dei vincoli di *trust*;

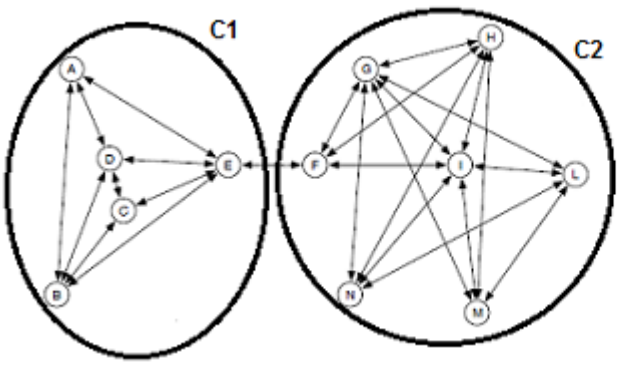
- un termine che approssimi il comportamento temporale di un peer nel sistema. Vogliamo infatti che i dati di un utente possano essere memorizzati sullo stesso nodo il più a lungo possibile, allo scopo di ridurre il traffico generato dall'abbandono del sistema da parte di una social cache che ha molti dati memorizzati. Come già discusso infatti, nel nostro modello un nodo che passa offline e memorizza i dati di un nodo offline dovrà passarli a un vicino prima di disconnettersi, così da garantire la persistenza di tali dati nella rete.

3.5 Il calcolo del Social Score

In questo paragrafo studieremo quali sono le caratteristiche che un nodo deve avere per essere un buon punto di memorizzazione per i dati dei nodi. In 3.5.1 ci occuperemo delle caratteristiche di centralità dei nodi mentre in 3.5.2 della disponibilità dei diversi utenti del sistema.

3.5.1 Misure di centralità nel calcolo del Social Score

[15] propone di utilizzare l'Ego Betweenness Centrality per individuare nodi con le caratteristiche strutturali più adatte ad assumere il ruolo di social cache. Tuttavia questa misura, come mostrato nell'Esempio 3.5.1, non sembra essere sempre quella che permette di individuare i nodi più adatti dal punto di vista strutturale a essere eletti social cache.



Nodi	BC	EBC	Deg	Triads	CC
A	0	0	3	3	1
B	0.33	0.33	4	5	0.83
C	0	0	3	3	1
D	0.33	0.33	4	5	0.83
E	28.33	4.33	5	5	0.5
F	30	3	4	3	0.5
G	7.5	1.67	6	10	0.67
H	4.25	1.0	5	7	0.7
I	7.5	1.67	6	10	0.67
L	0.25	0.33	4	5	0.83
M	0.25	0.33	4	5	0.83
N	0.25	0.33	4	5	0.83

Figura 3.6: Misure di centralità nel calcolo del Social Score

Esempio 3.5.1. Consideriamo il grafo di Figura 3.6 con le relative misure riportate nella tabella a fianco. I nodi che hanno la BC e la EBC più alta sono ovviamente E e F che mettono in comunicazione i due grandi cluster indicati con C1 e C2: questi nodi

quindi sono molto importanti per la diffusione dell'informazione all'interno della rete. Tuttavia essi non possono essere considerati per questo motivo buone social cache: il fatto che questi nodi si trovino su molti cammini minimi all'interno della propria ego network risulta infatti piuttosto irrilevante. Supponiamo infatti che F venga scelto come punto di memorizzazione dei dati. Esso memorizzerà o potrà accedere ai dati dei nodi E, G, H e I nel caso passassero offline. Ma chi potrà accedere a tali dati? Esattamente gli stessi nodi, ovvero gli amici di F. E, ad esempio, non sarà mai interessato ai dati dei nodi del cluster C2, dato che non presenta con loro alcun legame di amicizia. Inoltre i dati di G, H e I, se memorizzati in F, non saranno accessibili direttamente attraverso esso ai nodi L, M e N che però potrebbero essere interessati a essi al momento in cui uno tra i nodi G, H e I passerà offline. □

Abbiamo quindi osservato come la EBC in alcune situazioni non selezioni nodi particolarmente adatti a svolgere il ruolo di social cache. Vediamo quali misure possono essere usate in alternativa, tenendo presente che siamo interessati a nodi che hanno molti amici (grado alto) e che hanno molti amici che sono a loro volta amici tra loro. Immaginiamo infatti un nodo n che ha due vicini $v1$ e $v2$ che sono a loro volta vicini tra loro. Se n fosse eletto social cache, $v2$ potrebbe accedere ai dati di $v1$ tramite n quando $v1$ è offline e viceversa. Se il legame tra $v1$ e $v2$ non esistesse invece n memorizzerebbe o sarebbe in grado di accedere ancora ai dati dei due nodi, ma nessuno di essi sarebbe in realtà interessato ai dati dell'altro.

Cerchiamo quindi nel grafo nodi caratterizzati, oltre che da un alto grado (il che indica che il nodo ha molti amici e coprirebbe gli archi verso di essi in caso di una loro disconnessione), da un alto numero di triangoli (o *transitive triad*) centrati in tale nodo. Quest'ultima misura influenza direttamente il valore del coefficiente di clustering, che può essere ridefinito come $CC(n) = \frac{TransitiveTriads}{\binom{deg(n)}{2}}$.

Proponiamo dunque una nuova misura di Social Score definita come segue:

$$SS = (deg + TransitiveTriads) \cdot NodeAvailability \tag{3.2}$$

dove *deg* è il grado del nodo, $TransitiveTriads$ il numero di triadi transitive centrate in esso e $NodeAvailability$ una misura della disponibilità online del nodo che ci proponiamo di raffinare in seguito. L'introduzione del segno $\cdot$ tra i due fattori della formula serve unicamente ad attribuire loro lo stesso peso senza bisogno di normalizzarli.

Esempio 3.5.2. Esaminiamo ancora il grafo di Figura 3.6. Consideriamo una configurazione iniziale in cui tutti i nodi sono online e supponiamo si vogliano individuare dei buoni punti di memorizzazione per l'intera rete attraverso un approccio distribuito.

Supponiamo che, come per l'algoritmo del Paragrafo 2.5.3, ogni nodo selezioni come social cache un nodo della propria ego network in base al punteggio di Social Score. Utilizzando la nostra misura di Social Score nel cluster C1, a parità di *NodeAvailability*, tutti i nodi eleggono il peer E come social cache: A e C hanno infatti punteggio di Social Score pari a 6, B e D pari a 9, ed E pari a 10. E ha infatti grado più alto e può essere quindi eletto come social cache da un nodo in più, cioè F. Nel cluster C2 invece il nodo F ha Social Score pari a 7, L, M e N pari a 9, H pari a 12 e G e I pari a 16. Tutti i nodi eleggerebbero il nodo tra G e I con la *NodeAvailability* più alta (con l'ovvia esclusione di G o I stesso che eleggerebbe l'altro). Entrambi questi nodi coprirebbero tutti gli archi del cluster in caso di disconnessione di qualsiasi nodo e potrebbero essere utilizzati da tutti i nodi per accedere ai dati di tutti i loro vicini nel cluster. Questo è anche il caso in cui viene esemplificato cosa intendiamo per "potenziali" buoni punti di memorizzazione dei dati. Anche se entrambi sono social cache, non ha alcun senso memorizzare i dati di un nodo offline sia su G che su I: se essi sono memorizzati su G, I può accedervi chiedendoli direttamente a G che è suo amico.

Osserviamo infine che F risulta essere addirittura la scelta peggiore: se fosse scelto i nodi L, M e N non potrebbero comunque accedere direttamente tramite esso ai dati di G, H e I, dato che F non è un loro vicino. Ciò è evidenziato dal basso grado di tale nodo. □

In definitiva la nostra nuova formulazione del punteggio di Social Score sembra in grado di selezionare buoni punti di memorizzazione globali per l'intera rete.

Consideriamo ora il problema dell'individuazione di un punto di memorizzazione locale per memorizzare i dati di uno specifico nodo n. Le caratteristiche di centralità del nodo da selezionare sono diverse rispetto al caso precedente, in cui si cercava una nozione globale di centralità che consentisse di individuare un nodo dove tutti i vicini potessero memorizzare i propri dati. Il nodo non dovrà essere più centrale nell'intera rete, ma solo nella ego network di n: solo gli amici di n sono infatti interessati ad accedere ai suoi dati. La misura di centralità di un nodo m nella ego network e di un suo vicino n è semplicemente data dal grado locale di m in e. Questo valore indica infatti il numero di nodi che possono accedere ai dati di n tramite m nei momenti in cui n è offline. Definiamo quindi il Social Score locale di un nodo come:

$$SS_{ego_n} = deg_{loc_n} \cdot NodeAvailability \tag{3.3}$$

dove con deg_{loc_n} indichiamo il grado locale di un nodo nell'ego network di n. Tale valore è pari al numero di amici comuni che il nodo ha con n.

Esempio 3.5.3. Consideriamo l'ego network del nodo X del grafo di Figura 3.7.

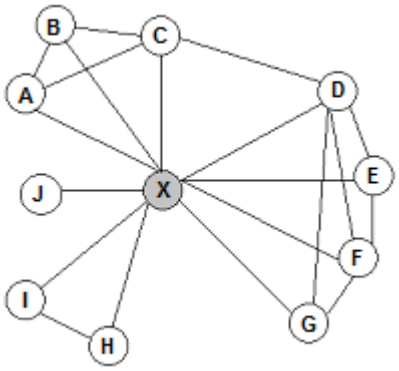


Figura 3.7: Grado locale dei nodi

I gradi locali risultanti sono $deg_{loc_X}(J) = 0, deg_{loc_X}(H) = deg_{loc_X}(I) = 1, deg_{loc_X}(A) = deg_{loc_X}(E) = deg_{loc_X}(G) = 2, deg_{loc_X}(B) = deg_{loc_X}(C) = deg_{loc_X}(F) = 3, deg_{loc_X}(D) = 4$. Il nodo D risulta pertanto il miglior punto di memorizzazione locale per i dati di X, poiché anche i nodi C, E, F e G potranno accedere direttamente a essi tramite tale nodo nei momenti in cui X si trova offline. □

Il seguente esempio mostra perché il Social Score locale è una misura di centralità più adatta per individuare un punto di memorizzazione locale di un nodo.

Esempio 3.5.4. Consideriamo il grafo di Figura 3.8.

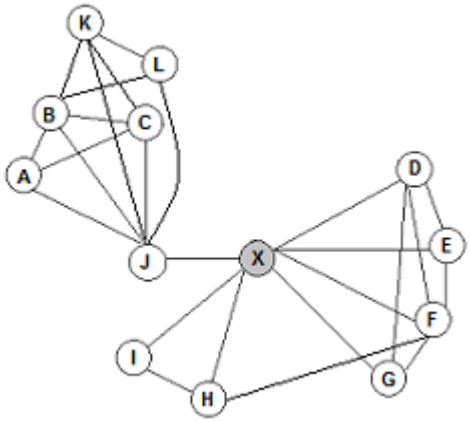


Figura 3.8: Social Score locale vs Social Score globale

Il nodo con Social Score (globale) più alto nella ego network di X è J, ma se X passasse a tale nodo i suoi dati nessuno dei suoi vicini, unici nodi interessati ad essi, potrebbero accedere direttamente ai dati. In un simile scenario è molto più vantaggioso affidarli al nodo con grado locale più alto, cioè F, così che anche D, E, G e H possano accedere ai dati di X tramite esso. □

3.5.2 Misure di *availability* nel calcolo del Social Score

L'algoritmo descritto nel Paragrafo 2.5.3 è una delle prime proposte che introduce una misura del comportamento temporale di un nodo nel punteggio di Social Score. In particolare la formula utilizza la $NodeAvailability$, definita come $NodeAvailability = \frac{TempoMedioOnline}{TempoMedioOffline}$ per descrivere i nodi caratterizzati da maggior disponibilità all'interno della rete. Il raffinamento di tale formula è uno dei principali obiettivi di questa tesi.

La $NodeAvailability$ risulta infatti essere una misura troppo grossolana per caratterizzare il comportamento temporale di un peer. Consideriamo ad esempio un utente che sta in media online 3 ore al giorno e che avrà quindi una $NodeAvailability$ di $\frac{3}{24-3} = \frac{1}{7} \simeq 0,14$. Se tale nodo sta generalmente online per 3 ore consecutive al giorno sarà certamente un buon candidato a memorizzare i dati dei suoi vicini in tale fascia temporale. Se invece, pur rimanendo in media giornalmente online per 3 ore, ha un comportamento intermittente, alternando brevi periodi online con periodi offline, esso non costituisce un buon candidato, perché ad ogni sua disconnessione le informazioni in esso memorizzate, dovranno essere passate ad altri nodi, con un aumento sostanziale di traffico nella rete. Il nostro obiettivo è quindi quello di *mantenere le social cache stabili nel tempo*. A tal proposito il nodo che, dal solo punto di vista temporale, è il miglior candidato ad assumere il ruolo di social cache per un dato peer è quello, tra i nodi della sua ego network connessi in quel momento, che rimarrà online più a lungo. Risulta necessaria quindi un'analisi dei periodi online e offline dei nodi di una OSN, in modo da capire quali siano i fattori che meglio caratterizzano il suo comportamento temporale. A questo scopo abbiamo effettuato un'analisi approfondita sulla base di alcuni dataset reali. Riportiamo nel Paragrafo 3.5.3 solo le conclusioni utili per definire la nostra misura di Social Score e rimandiamo al Paragrafo 5.1 per una loro discussione più approfondita.

3.5.3 Availability-pattern in OSNs

L'analisi dei periodi online e offline dei nodi è stato effettuata mediante lo studio di tre dataset contenenti informazioni sullo stato di connettività dei peer nell'arco di circa 40 giorni in tre diverse OSNs: Tagged, Netlog e MySpace.

Il nostro primo tentativo, sulla base di diversi studi effettuati sul problema della replicazione in sistemi distribuiti ([7], [37]) è stato quello di analizzare in che percentuale le connessioni dei peer fossero periodiche e se fosse possibile predire i periodi di connettività dei peer sulla base del loro comportamento passato. Abbiamo allo scopo creato dei predittori che tentassero di predire la presenza online di un peer in un determinato intervallo temporale sulla base del fatto che lo stesso peer fosse online o meno nello stesso intervallo di tempo nei giorni precedenti. Purtroppo i risultati, a differenza di quelli ottenuti in altri lavori per sistemi di file-sharing (come ad esempio [7]), sono stati negativi. Le connessioni dei peer risultano essere in gran parte aperiodiche e difficilmente predicibili. Pur rilevando comportamenti periodici ripetuti sull'intera rete (ad esempio una variazione ciclica del numero dei peer sulla base dell'alternarsi della notte e del giorno) è risultato impossibile anche estrarre un sottoinsieme di peer con caratteristiche di alta predicibilità. Sono emersi però diversi fattori interessanti: innanzitutto la difficoltà nel predire il comportamento dei peer sembra in parte da attribuirsi allo scarso tempo che essi trascorrono online (circa un'ora di media al giorno), mentre i sistemi di file-sharing sono caratterizzati da una presenza online molto più elevata.

Abbiamo infine osservato come sia possibile stimare il comportamento temporale di un peer online attraverso le caratteristiche delle sue sessioni. La durata delle sessioni risulta essere una peculiarità del comportamento dei singoli utenti: esistono gruppi di utenti con durata media delle sessioni molto diverse tra loro, e una buona percentuale dei peer ha sessioni decisamente più lunghe della media. Ogni utente inoltre presenta sessioni piuttosto omogenee nel tempo. Vogliamo quindi introdurre un valore $EstimatedUptime$ che stimi il tempo che ci attendiamo un peer resti online data la sua presenza nel sistema al momento corrente. Ci proponiamo quindi di utilizzare tale valore nella formula per il calcolo del Social Score per stimare quale dei peer abbiano un comportamento temporale più adatto per assumere il ruolo di social cache.

3.5.4 Sessioni degli utenti e formula Social Score

Applichiamo le conclusioni trovate al problema del social caching. Nel primo algoritmo proposto avevamo utilizzato la $NodeAvailability$, definita come $NodeAvailability = \frac{TempoMedioOnline}{TempoMedioOffline}$, come misura del comportamento temporale di un nodo. Questa misura, pur molto semplice, risulta essere un'approssimazione accettabile in un sistema senza rielezione delle social cache: i dati analizzati mostrano che i peer stanno in media online solo un'ora al giorno e quindi un nodo che sta molto online coprirà certamente una parte dei periodi offline di una buona frazione dei propri vicini.

Le cose cambiano se, come più naturale, vogliamo introdurre un meccanismo di rielezione delle social cache quando queste passano offline, in modo da garantire comunque la persistenza delle informazioni in esse contenute. In tale contesto i nodi con un comportamento temporale più adatto a essere scelti come punti di memorizzazione sono, in ogni istante, quelli che si prevede rimanere consecutivamente online per il più lungo lasso di tempo: in questo modo infatti è possibile mantenere i dati memorizzati nello stesso nodo per il periodo più lungo possibile. La nostra proposta è quella di stimare il tempo che un peer resterà online con la durata media delle sue sessioni. La durata delle sessioni non è ovviamente costante per buona parte dei peer, tuttavia la loro durata media, data la diversità osservata tra diversi gruppi di peer, sembra fornirci comunque un'indicazione attendibile. Ciò è rafforzato dall'osservazione che, quando andiamo a eleggere un punto di memorizzazione, scegliamo il nodo da selezionare tra quelli online. La probabilità di trovare un peer online in una sessione più breve della media è tanto più bassa tanto più breve è la durata di tale sessione: in altre parole la possibilità di sovrastimare la durata della sessione corrente di un peer utilizzando il suo valore medio certamente esiste, ma ciò non accade frequentemente dato che la probabilità di trovare un peer online in una data sessione è tanto maggiore tanto più lunga è la durata della sessione stessa. Riteniamo quindi che la durata media delle sessioni di un peer sia un buon fattore per stimare il tempo che un peer attualmente connesso trascorrerà online.

In definitiva per stimare il tempo che un nodo rimarrà online nella rete possiamo utilizzare il valore

$$EstimatedUptime = \max(MediumSessionLength - CurrentUptime, 1) \qquad (3.4)$$

dove $MediumSessionLength$ è la durata media delle sessioni del peer e $CurrentUptime$ è la durata attuale della sessione corrente, cioè da quanto tempo il peer si trova online. L'introduzione della funzione massimo serve solo a garantire che il Social Score sia sempre maggiore di 0. Per far ciò attribuiamo un valore minimo di $EstimatedUptime$ pari a 1 minuto, facendo quindi l'assunzione che anche un peer che è online da un tempo maggiore della sua sessione media abbia buone probabilità di rimanere online per almeno un minuto prima di disconnettersi.

Se immaginiamo che un nodo memorizzi i propri dati in un vicino fino al momento in cui esso va offline, dopodiché viene scelto un nuovo punto di memorizzazione, il nodo della ego network con il valore di $EstimatedUptime$ più alto risulta essere quello con il comportamento temporale più adatto a diventare un nuovo punto di memorizzazione.

Proponiamo quindi di utilizzare l'$EstimatedUptime$ come misura della disponibilità di un nodo per il calcolo del Social Score. La formula 3.2 diventa quindi:

$$SS = (deg + TransitiveTriads) \cdot EstimatedUptime \tag{3.5}$$

mentre quella per il calcolo del Social Score locale diventa:

$$SS_{ego_n} = deg_{loc_n} \cdot EstimatedUptime \tag{3.6}$$

3.6 Algoritmo per l'elezione di Punti di Memorizzazione

Vediamo ora come sia possibile garantire che i dati di un peer, sia esso online o offline, siano sempre memorizzati su almeno un altro peer. Garantiamo cioè che i dati di ogni peer p siano memorizzati in ogni momento in un punto di memorizzazione locale (PML_p) appartenente alla sua ego network. Il PML_p verrà scelto tra le social cache sulla base del punteggio di Social Score locale SS_{ego}.

Presentiamo quindi l'algoritmo per l'elezione di un buon punto di memorizzazione per un nodo. L'algoritmo viene eseguito unicamente da un peer p che riceve una notifica di disconnessione dal nodo che costituisce il suo PML: in questo caso p deve rieleggere un nuovo punto di memorizzazione per i propri dati. Il peer, per non aumentare il numero di social cache elette, cercherà di scegliere un nodo tra i propri vicini che sono già social cache sulla base del punteggio di Social Score locale. Se tra di essi ce n'è almeno uno con un valore del punteggio maggiore di una soglia base lo sceglierà come proprio punto di memorizzazione locale, altrimenti eleggerà una nuova social cache, cioè un nuovo punto di memorizzazione globale per la rete. Questa sarà il proprio vicino con punteggio di Social Score (globale) massimo e si occuperà di notificare la propria elezione a tutti i suoi vicini online. Naturalmente può accadere che tale nodo sia già social cache: in questo caso non verrà scelto alcun nuovo punto di memorizzazione globale. L'algoritmo descritto è mostrato in Algorithm 7.

Nell'algoritmo N_p è l'insieme dei vicini online di un peer p, PML_p indica il punto di memorizzazione locale di p, $isSocialCache()$ è un metodo che restituisce $true$ se il nodo che l'invoca è social cache e $false$ altrimenti, $BLACK$ è un'etichetta che indica che il nodo è stato scelto come social cache e λ_{min} rappresenta il valore minimo del punteggio di Social Score locale che una delle social cache già elette deve avere per poter diventare il punto di memorizzazione locale del nodo elettore; altrimenti verrà eletta una nuova social cache. Tale soglia determina le caratteristiche dei punti di memorizzazione cercati:

Algorithm 7 Elezione Punto Memorizzazione

if $N_p \neq \varnothing$ **then**
  get $SS_{ego_p}(v)\ \forall v \in N_p |\ v.isSocialCache() = true$;
  select $n \mid SS_{ego_p}(n) = \max\{SS_{ego_p}\}$;
  if $SS_{ego_p}(n) > \lambda_{min}$ **then**
    $PML_p = n$;
  else ▷ Selezione di un nuovo punto di memorizzazione globale
    get $SS_v\ \forall v \in N_p$;
    select $n \mid SS_n = \max_{v \in N(p)}$;
    if $notBLACK$ **then** ▷ Se non è una social cache lo diventa
      mark n as $BLACK$;
    end if
    $PML_p = n$;
  end if
end if

se scegliamo per esso un valore molto alto un nodo selezionerà sempre il miglior punto di memorizzazione tra i propri vicini al prezzo di aumentare il numero delle social cache, se scegliamo un valore basso potremmo scegliere un punto di memorizzazione peggiore ma ridurremo il numero di social cache totali nel sistema.

3.7 Dinamicità delle social cache e algoritmo di copertura

Occupiamoci ora di garantire che ogni nodo possa accedere ai dati dei propri vicini offline. Questo, come anticipato, viene garantito tramite un algoritmo di copertura, il cui significato è stato spiegato nel Paragrafo 3.3. Supponiamo che un nodo al momento di passare offline riesca sempre, se necessario, a passare i dati da esso memorizzati a un proprio vicino prima di disconnettersi. Nel nostro modello consideriamo infatti solo abbandoni volontari del sistema, mentre non trattiamo disconnessioni dovute a fallimenti. Per quest'ultimo caso assicuriamo soltanto la presenza di una copia dei dati nel sistema, contenuta nel punto di memorizzazione locale del nodo.

Consideriamo inizialmente una configurazione della rete in cui i nostri vincoli sono soddisfatti: ogni nodo (online o offline) memorizza i propri dati in un punto di memorizzazione locale e ciascun peer può accedere ai dati dei propri vicini offline tramite una social cache; tutti gli archi sono quindi coperti, secondo la nostra nozione di copertura. I peer della rete possono essere divisi in due gruppi:

- nodi che non posseggono dati di altri peer: questi sono i nodi che non sono social cache ma anche le social cache che non sono un PML per i dati di alcun nodo (vedi

Esempio 3.3.1);

- nodi che posseggono dati di altri peer: questi sono i peer che costituiscono un PML per i dati di qualche utente e le social cache elette dall'algoritmo di copertura che stiamo descrivendo che, per la politica di distribuzione dei dati tra le social cache (che non è stata studiata in questa tesi), posseggono comunque i dati di alcuni peer.

Supponiamo che un nodo n vada offline. Abbiamo due casi diversi da trattare, corrispondenti ai due tipi di nodi precedenti.

Nodi che non posseggono dati di altri peer

Consideriamo il caso in cui un nodo n vada offline: è necessario garantire la copertura degli archi verso n, ovvero che i vicini di n siano in grado di accedere ai suoi dati. n informerà i propri vicini della sua disconnessione e questi verificheranno se tutti i propri archi uscenti sono coperti da una social cache o meno. Chiameremo $outStar$, per *outgoing star*, cioè stella uscente, l'insieme degli archi uscenti di un nodo che non sono coperti. I nodi che hanno tutti gli archi uscenti coperti ($outStar = 0$) non dovranno far nulla, perché sono in grado di accedere ai dati del nodo che sta andando offline, quelli che hanno almeno un arco scoperto avvieranno invece una procedura di copertura degli archi. Questa si basa sullo span dei nodi, ovvero sul numero di archi scoperti nelle loro ego network. Ogni nodo eleggerà nella propria ego network il nodo, tra quelli che non sono social cache, con il valore dello *span* maggiore come nuova social cache. Questi nodi infatti con la loro elezione copriranno anche archi che fanno parte delle ego network di altri nodi. Dopo questo primo passo se $outStar = 0$ la procedura termina, altrimenti lo *span* dei vicini viene ricalcolato e si itera. Al termine della procedura tutti i nodi avranno tutti gli archi uscenti coperti e i nodi eletti social cache notificheranno ai vicini la propria elezione.

Questa procedura realizza la copertura degli archi ed è mostrata, per un generico nodo n, in Algorithm 8. Esso viene eseguito da un nodo che riceve una notifica di disconnessione da parte di un vicino in seguito alla quale ha almeno un arco della sua stella uscente scoperto.

Nell'algoritmo $isSocialCache()$ e $BLACK$ hanno lo stesso significato che avevano in Algorithm 7. Ricordiamo infine che lo *span* di un nodo è il numero di archi scoperti nella sua ego network e che un arco eletto social cache copre tutti gli archi offline della propria ego network.

L'Esempio 3.7.1 mostra l'esecuzione dell'algoritmo sul grafo di Figura 3.9.

Algorithm 8 Algoritmo copertura per un nodo n

```
1:  function OnMessageReceive(disconnect(p))
2:      if outStar(n) = 0 then
3:          return;
4:      else
5:          while outStar(n) > 0 do
6:              get span(v)  ∀v ∈ N_n|!(v.isSocialCache());
7:              select m | span(m) = max{span(v)};
8:              compute span(n);
9:              if span(n) ≥ span(m) then
10:                 mark n as BLACK;
11:                 inform every node in N_n;
12:                 return;
13:             else
14:                 mark m as BLACK;
15:             end if
16:         end while
17:     end if
18: end function
```

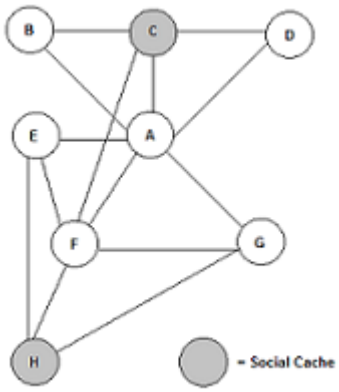


Figura 3.9: Algoritmo di copertura

Esempio 3.7.1. Consideriamo una configurazione iniziale in cui tutti i nodi sono online. Ogni peer deve avere una social cache nella propria ego network che costituisca il suo PML. Supponiamo quindi che il nodo A vada offline. La disconnessione viene notificata ai nodi B, C, D, E, F e G. I nodi B, C, D hanno l'arco verso A (l'unico che può essere scoperto) coperto. Il nodo C infatti, che è una social cache ed è un amico di B e D possiede o può accedere ai dati di A. E, F, G hanno il rispettivo arco verso A scoperto ($outStar > 0$) e quindi eseguono l'algoritmo di copertura. E, F e G al primo ciclo chiederanno il valore dello span ai propri vicini online che non sono social cache. I valori dello span dei nodi sono $span(E) = 2, span(F) = 3$ e $span(G) = 2$. E e G chiedono a F il suo valore dello span, lo confrontano con il proprio ed eleggono F come social cache. F chiede il valore dello span a E e G, lo confronta con il proprio e elegge se stesso. Dopo il primo ciclo e con l'elezione di un solo nodo come social cache vengono coperti tutti e tre gli archi (E,A), (F,A) e (G,A): tutti e tre i nodi interrompono quindi l'esecuzione dell'algoritmo. Per curiosità sottolineiamo che in questo caso F non ha certamente bisogno di memorizzare i dati di A perché può accedere a essi tramite C. □

L'algoritmo viene eseguito in parallelo da tutti i nodi che hanno qualche arco della loro ego network scoperta: ciò può essere dovuto alla disconnessione di un vicino che era social cache o, come vedremo in seguito, alla riconnessione del nodo nel sistema.

Nodi che posseggono dati di altri peer

In questo caso si deve comunque eseguire il procedimento del caso precedente, perché potrebbero esserci comunque archi scoperti; inoltre si deve garantire che tutti i nodi di cui il peer che sta passando offline è PML eleggano un nuovo PML. Questi nodi possono essere online o offline. Se sono online riceveranno una notifica della disconnessione di n e eleggeranno un nuovo PML con Algorithm 7.

Dobbiamo quindi aggiungere questo caso agli algoritmi precedenti: se il nodo da cui un nodo n riceve la notifica di disconnessione era il suo PML, n deve procedere alla rielezione del proprio PML.

La procedura sopra descritta è implementata da Algorithm 9:

Nell'algoritmo $ElezionePuntoMemorizzazione()$ produce l'esecuzione di Algorithm 7. La funzione $copertura()$ corrisponde invece alle righe 2-16 di Algorithm 8.

Più complessa è la gestione dei dati di un vicino p offline di n quando n passa offline. n deve eleggere un PML per conto di p a cui passare i dati. Per mantenere i vincoli di *trust* il nodo dovrà essere scelto tra i vicini comuni con p che sono online e tra essi verrà scelta, allo scopo di minimizzare il numero di social cache elette, la social cache

Algorithm 9 Ricezione notifica di disconnessione

```
function OnMessageReceive(disconnect(p)) ▷ ricezione msg disconnessione nodo p
    if PML_n = p then
        ElezionePuntoMemorizzazione();
    end if
    n.copertura();
end function
```

con Social Score locale più alto. Se non esistono social cache tra i vicini comuni verrà scelto, sempre tra di essi, quello con Social Score (globale) più alto come nuova social cache e nuovo PML per i dati di p. Tale nodo notificherà la propria elezione ai vicini.

Il problema nasce nel caso, improbabile dato che scegliamo sempre punti di memorizzazione centrali nella rete, in cui non esistano vicini online comuni con p a cui passare i dati. Abbiamo in questo caso due soluzioni. La prima, più drastica, è di non garantire la persistenza dei dati di p nel sistema, l'altra è quella di rilasciare i vincoli di *trust*, chiedere ai vicini di individuare un nodo m della loro ego network che abbia come vicino p e passare poi a un tale vicino v i dati di p, in modo che li passi a m. Tale scenario è mostrato in Figura 3.10. In questo caso comunque v, che è un "amico di un amico" di p, farebbe solo "da tramite" per il passaggio dei dati e non li memorizzerebbe mai effettivamente. Chiameremo tale procedura, che verrà utilizzata solo nei casi strettamente indispensabili, $findNeighbourOfNeighbour()$.

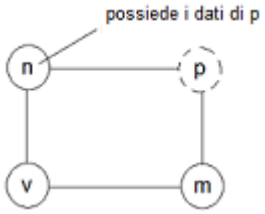


Figura 3.10: Scenario di esecuzione della procedura $findNeighbourOfNeighbour()$

Un'altra situazione estrema è quella in cui un nodo n che sta passando offline non ha nessun vicino online: in tale configurazione nessun nodo è interessato ai suoi dati, ma un vicino v di n che si riconnette al sistema prima di n stesso potrebbe voler accedere ai suoi dati. In questo caso una possibile soluzione potrebbe essere far scegliere all'utente come gestire i propri dati: l'utente potrebbe individuare un nodo fidato online a cui passare i propri dati tra gli amici degli amici o tra i propri amici che non rientrano tra i 150 vicini dell'overlay Dunbar-based oppure scegliere di non consentire a nessuno l'accesso

ai propri dati fino alla sua riconnessione nel sistema.

La procedura per la rielezione del PML di un nodo offline da parte di un nodo in fase di disconnessione che possiede i suoi dati è implementata da Algorithm 10.

Algorithm 10 Rielezione PML per nodi offline eseguito da nodo p in disconnessione

```
1: function OnDisconnect(p)                          ▷ disconnessione del nodo p
2:     if p.hasData() then
3:         for all u ∈ N(p)| !(u.isOnline()) ∧ PML_u = p do
4:             /*per tutti i nodi offline di cui p era PML*/
5:             get CN = CommonNeighbors(p, u);
6:             CN_sc = CN ∩ SocialCache;          ▷ amici comuni che sono social cache
7:             if CN_sc ≠ ∅ then
8:                 get SS_ego_u(y) ∀y ∈ CN_sc;
9:                 select w | SS_ego_u(w) = max_{j∈CN_sc}{SS_ego_u(j)};
10:                PML_u = w;
11:                send data_u to w;
12:            else
13:                if CN ≠ ∅ then
14:                    get SS_y ∀y ∈ CN;
15:                    select w | SS_w = max_{j∈CN}{SS_j};
16:                    PML_u = w;
17:                    send data_u to w;
18:                    if notBLACK then          ▷ Se non è una social cache lo diventa
19:                        mark w as BLACK;
20:                    end if
21:                else
22:                    v = findNeighbourOfNeighbour(n);
23:                    send data_u to v;
24:                end if
25:            end if
26:        end for
27:    end if
28: end function
```

Nell'algoritmo $hasData()$ restituisce $true$ se il nodo ha dati memorizzati e $false$ altrimenti, $isOnline()$ restituisce $true$ se il nodo è online, $false$ altrimenti e $CommonNeighbors(x, y)$ restituisce l'insieme dei vicini comuni tra x e y. Per ogni vicino offline si calcolano i vicini comuni online e quelli tra essi che sono social cache (linee 5-6). Se c'è almeno un vicino comune online che è social cache si affidano i dati a quello tra essi che ha punteggio di Social Score locale più alto (ll. 7-11). Altrimenti se c'è comunque almeno un vicino comune online si affidano i dati a quello tra essi che ha punteggio di Social

Score (globale) più alto, che viene eletto social cache (ll. 13-20). Altrimenti affidiamo i dati a un vicino di un vicino trovato con la procedura $findNeighbourOfNeighbour()$ (ll.22-23).

La comunicazione ai nodi offline dei loro nuovi PML verrà data loro non appena tornano online dai PML stessi.

Esempio 3.7.2. Vediamo un semplice esempio, riferito al grafo di Figura 3.11.

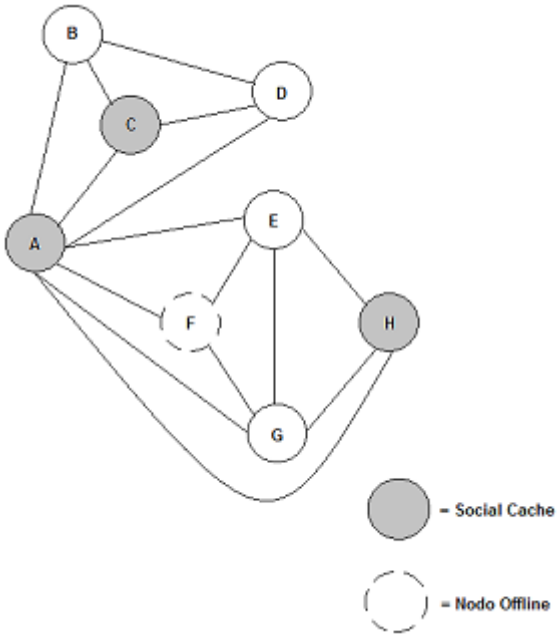


Figura 3.11: Rielezione PML per un nodo offline eseguito da un nodo in fase di disconnessione

Supponiamo che i nodi A, C e H siano social cache, e che i nodi abbiano i seguenti PML: $PML_A = PML_B = PML_D = C, PML_C = PML_F = PML_H = A, PML_E = PML_G = H$. Il nodo F è offline e i suoi archi verso A, E e G sono tutti coperti dalla social cache A. Supponiamo che la social cache A passi offline. Tutti i nodi dovranno far partire l'algoritmo di copertura, ma in questo caso fortunato tutti gli archi verso A risultano coperti e quindi non viene eletta nessuna nuova social cache. A è tuttavia PML dei nodi C, F e H. C e H, essendo online, si possono occupare direttamente dell'elezione di un nuovo PML, che verrà scelto sulla base del punteggio di Social Score tra B e D per C e tra E e G per H. Notiamo che in entrambi i casi non ci sono amici comuni che sono già social cache. A si deve invece occupare di trovare un nuovo PML per i dati di F.

A calcola $CommonNeighbors(A, F) = \{E, G\}$ e, dato che nessuno dei due nodi è social cache, passa i dati di F a quale di questi nodi ha il punteggio di Social Score (globale) più alto. Tale nodo sarà lo stesso scelto da H come proprio PML. □

Riconnessione di un nodo al sistema

Quando un nodo n torna online potrebbe avere un arco verso un amico offline o scoperto e potrebbe quindi non essere in grado di accedere ai dati di o. Abbiamo però la certezza che tutti gli amici comuni tra n e o che si trovano online sono in grado di accedere ai dati di o. n quindi può selezionare come social cache uno di tali vicini per poter a sua volta accedere ai dati di o. Anche in questo caso c'è però uno scenario sfortunato (e molto raro in contesti realistici) in cui non è possibile garantire allo stesso tempo l'accesso di n ai dati di o e il soddisfacimento dei vincoli di *trust*, che è quello in cui non esistono amici comuni online tra n e o. In questo caso i dati di o sono certamente memorizzati nel PML di o ma tale nodo non è amico di n. Come nello scenario precedente in questo caso si può ricorrere ad un amico comune tra n e il PML di o che faccia solo "da tramite" per il passaggio dei dati (senza memorizzarli effettivamente) in modo che n possa accedervi.

Un nodo n che torna online dovrà quindi eseguire un algoritmo di copertura del tutto simile a quello di Algorithm 8, con l'unica differenza che se non esistono amici comuni online con un amico offline o si deve procedere al recupero (che può avvenire anche *on-demand*) dei dati di n dal suo PML.

Capitolo 4

Implementazione del sistema

Questo capitolo presenta la parte implementativa di questa tesi di laurea. I protocolli presentati nel capitolo precedente vengono implementati mediante PeerSim, un simulatore di reti P2P in grado di simulare l'esecuzione di protocolli su un numero relativamente alto di nodi. Nel Paragrafo 4.1 vengono discusse le caratteristiche più interessanti di PeerSim; viene quindi presentata l'architettura complessiva della simulazione (Paragrafo 4.2). Si mostra poi come è stato implementato l'overlay sociale Dunbar-based: l'implementazione dei nodi viene discussa nel Paragrafo 4.3 e quella delle Social Table nel Paragrafo 4.4. Il Paragrafo 4.5 discute l'implementazione di un modello di Churn che riesce a descrivere in maniera realistica le sessioni dei nodi facenti parte della rete. Infine nel Paragrafo 4.6 discutiamo in dettaglio le caratteristiche del file di configurazione, che definisce le impostazioni necessarie all'esecuzione di tutte le componenti della simulazione.

4.1 PeerSim: un simulatore P2P scalabile

La valutazione di un protocollo o di un algoritmo per un sistema P2P non è generalmente banale. Un insieme di fattori, tra cui la necessità di disporre di un alto numero di host, la loro dinamicità, la ripetibilità degli esperimenti, etc., rende di fatto impossibile costruire ambienti reali dove testare tali protocolli. Una delle soluzioni maggiormente impiegate dai ricercatori del settore è quindi quella di utilizzare simulatori altamente scalabili.

Per testare gli algoritmi e i protocolli che abbiamo proposto nel Capitolo 3 utilizzeremo PeerSim [38], un simulatore open source basato su Java e sviluppato all'Università di Bologna. PeerSim ha caratteristiche di grande scalabilità e configurabilità: ogni simulazione è infatti definita da un insieme di classi "core" a cui si aggiungono un insieme

di componenti definite dall'utente che possono essere "inserite" nel simulatore. I riferimenti a tali componenti sono raccolti in un file di configurazione e vengono caricate dinamicamente dal simulatore. La simulazione è completamente sequenziale a thread unico, così che un'unica macchina è in grado di simulare l'intera rete P2P.

PeerSim fornisce due modalità di simulazione diverse: nel modello cycle-driven i nodi che fanno parte del sistema vengono considerati in sequenza e su ognuno di essi si eseguono tutti i protocolli. Lo scambio di messaggi tra i nodi è simulato mediante l'esecuzione di metodi, mentre non è presente la simulazione del livello di trasporto dell'overlay. La modalità di simulazione event-driven è più realistica. Lo scambio di messaggi nella rete è modellato attraverso protocolli send-receive e l'esecuzione dei protocolli presenti su ogni nodo è guidata dagli eventi, cioè dalla schedulazione dei messaggi ricevuti dai nodi. L'avanzamento del tempo della simulazione può avvenire secondo due diverse modalità. Nella discrete-event simulation esso avviene ad incrementi costanti pari a un valore prefissato. Nella modalità event-driven invece ogni evento ha un timestamp associato e il tempo "salta" da un timestamp al successivo senza che i periodi tra uno e l'altro abbiano alcun significato. Per generare un nuovo evento occorre definirlo specificando il suo delay rispetto al tempo attuale, l'oggetto che descrive l'evento, il nodo a cui è destinato e il protocollo, tra quelli in esecuzione sul nodo, che riceverà l'evento. Gli eventi sono organizzati in una coda coordinata dal simulatore e sono gestiti mediante un event-handler che contiene il metodo `processEvent()`: invocato su un nodo esso contiene le azioni da eseguire al momento della ricezione di un evento.

In PeerSim la rete è modellata come un vettore di nodi virtuali, creata clonando un nodo prototipo per un numero di volte pari alla dimensione della rete. Ogni nodo, che è sempre accessibile per il simulatore e raggiungibile dagli eventi generati da PeerSim, è identificato da un ID unico e contiene uno stack di protocolli: tutti i nodi eseguono gli stessi protocolli. Ogni evento è caratterizzato da un istante in cui viene ricevuto dal destinatario, il quale provvede a gestire il messaggio tramite un'apposita procedura.

Ogni nodo implementa l'interfaccia `Node` che contiene lo stato, l'insieme di protocolli eseguiti ed è caratterizzato da tre interfacce principali che ne determinano il comportamento: `Protocol, Linkable` e `Transport`.

L'interfaccia `Protocol` è molto generale e contiene un metodo `clone()` per allocare per ogni nodo copie diverse delle strutture dati accedute dal protocollo. Ogni nodo può avere molti elementi che implementano l'interfaccia `Protocol`: il corpo di questi metodi rappresenta l'insieme delle azioni che consentono la gestione dell'evento. L'interfaccia `Linkable` è un contenitore di metodi e protocolli: in essa si trovano i metodi per gestire la vista di un peer e i collegamenti con gli altri nodi della rete. L'interfaccia `Transport`

definisce il sistema di comunicazione usato dai nodi: il metodo `send()` simula l'invio di un messaggio a livello fisico. Il servizio di trasporto può essere affidabile o meno e si può associare una latenza variabile alla trasmissione dei messaggi.

Un'altra interfaccia di fondamentale importanza nella simulazione è l'interfaccia `Control`, che permette di definire operazioni che richiedono una conoscenza globale della rete. I controlli possono accedere a qualsiasi componente della rete e sono generalmente utilizzati per raccogliere le statistiche che riguardano la simulazione. Il controllo può essere di uno dei seguenti tipi:

- `Initializers:` sono controlli eseguiti all'inizio della simulazione per definire la topologia dell'overlay e lo stato iniziale dei nodi;
- `Dynamics:` sono controlli eseguiti periodicamente durante la simulazione ad esempio per aggiungere o rimuovere nodi dall'overlay;
- `Observers:` sono controlli eseguiti periodicamente durante la simulazione per registrare statistiche sullo stato dei nodi o, ad esempio, per calcolare l'aggregazione dei valori dei diversi nodi (grado medio, coefficiente di clustering, etc.).

Il simulatore permette all'utente di scegliere tra la modalità di simulazione cycle-based e quella event-based. È infine possibile effettuare simulazioni di tipo ibrido.

Come accennato in precedenza in PeerSim una simulazione viene specificata mediante il file di configurazione, un file di testo contenente i parametri globali della rete e l'insieme di protocolli e controlli che compongono la simulazione. Per eseguire la simulazione si deve digitare da linea di comando:

Listing 4.1: Esecuzione simulazione

```
java -cp <class-path>  peersim.Simulator ConfigurationFile.txt
```

dove `ConfigurationFile.txt` è il nome del file di configurazione.

4.2 Architettura della simulazione

La social overlay Dunbar-based e gli algoritmi introdotti nel Capitolo 3 vengono analizzati e valutati attraverso una simulazione implementata mediante PeerSim. La social overlay, costruita sulla base di un dataset reale contenente le frequenze di contatto tra utenti in Facebook, viene utilizzata come topologia base su cui osservare i protocolli definiti. L'implementazione con PeerSim porta a una suddivisione delle funzionalità

del sistema in un insieme di classi `Java` che implementano l'interfaccia `Protocol` o l'interfaccia `Control`.

In Figura 4.1 viene mostrata l'architettura della simulazione in termini di classi `Java`.

Il protocollo `ChurnModel` modella le sessioni degli utenti nella rete secondo il modello di Churn che verrà descritto nel Paragrafo 4.5. Ogni nodo possiede un'istanza di questo protocollo caratterizzato da parametri diversi che definiscono la durata e la frequenza delle sessioni. L'obiettivo del protocollo è avere utenti con caratteristiche di connettività diverse e tanto più simili possibile a quelle osservate nelle OSNs analizzate nel Capitolo 5.

Il livello `Linkable` definisce la topologia della rete attraverso le tabelle di routing: queste contengono tutte le informazioni necessarie agli utenti per mantenere i contatti verso i nodi a cui sono connessi. La `Social Table` contiene inoltre tre campi utili a gestire l'elezione delle social cache: il campo `EtichettaNodo` permette di sapere se il nodo è social cache o no, il campo `FaseElez` contiene lo stato dell'istanza del protocollo di elezione del nodo (il campo vale 0 se il nodo non sta eseguendo il protocollo, 1 se sta rieleggendo il proprio punto di memorizzazione locale e 2 se è in fase di disconnessione e deve rieleggere il punto di memorizzazione di qualche vicino offline di cui memorizzava i dati) e il campo `FaseCop` contiene lo stato dell'istanza del protocollo di copertura del nodo (il campo vale 0 se il nodo non sta eseguendo il protocollo, 1 se sta eseguendo la copertura).

Il livello `Initializer` definisce gli inizializzatori del protocollo Churn Model e della topologia di rete. Il primo setta lo stato iniziale di tutti i nodi a `Online`: questo è fatto per avere una configurazione iniziale della rete in cui tutti i nodi sono connessi. La classe `WireFromDatasetEgo` serve invece per inizializzare le Social Table di un numero ristretto di nodi estratti da un dataset reale ottenuto da una Facebook Regional Network. Per motivi di scalabilità purtroppo siamo costretti a lavorare con PeerSim con un numero limitato di nodi. Gli esperimenti fatti, riportati e discussi nel Paragrafo 5.2, indicano la validità del nostro approccio, che dovrà comunque essere valutato più opportunamente su reti con caratteristiche più simili a quelle reali.

Il livello `Controller` contiene le componenti che permettono di effettuare modifiche o osservazioni periodiche sullo stato globale del sistema o di sue singole componenti. I controller `ControlChurnEvolution`, `ControlCopertura` e `ControlElezione` regolano l'esecuzione dei rispettivi protocolli. In particolare il secondo è responsabile della corretta esecuzione del protocollo di copertura: grazie a esso è possibile simulare un'esecuzione distribuita dell'algoritmo complessivo. Gli altri controlli servono ad analizzare i protocolli di elezione e copertura (`NetworkSocialCacheObserver`), a raccogliere in-

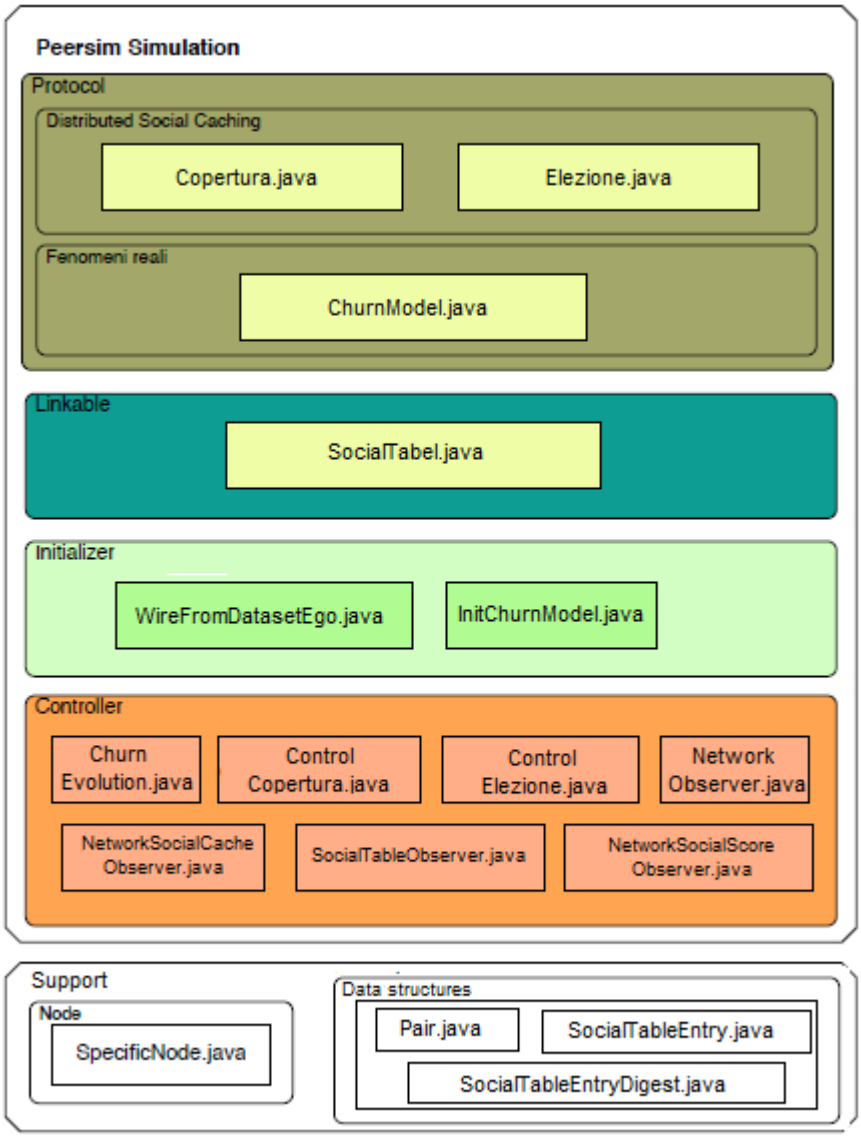


Figura 4.1: Diagramma delle classi

formazioni sul punteggio di Social Score dei nodi (`NetworkSocialScoreObserver)` e a osservare l'evoluzione della social overlay e delle Social Table dei nodi (`NetworkObserver` e `SocialTableObserver`).

Infine è stato creato un livello `Support` di supporto alla simulazione. La classe `SpecificNode.java` definisce gli elementi di base a disposizione della simulazione (cioè i Nodi), mentre le `Data Structures` contengono le strutture dati di supporto utilizzate all'interno delle Social Table, come una coppia generica (`Pair.java`) o il contenuto di una entry della Social Table (`SocialTableEntry.java`): quest'ultima struttura dati verrà analizzata approfonditamente nel Paragrafo 4.4.

4.3 Nodi

I nodi della simulazione sono definiti attraverso la classe `SpecificNode`, che implementa l'interfaccia predefinita `Node`. Il numero di nodi creato durante la simulazione è definito nel file di configurazione discusso nel Paragrafo 4.6. Ad ogni nodo è associato un identificatore unico, che rimane uguale per tutta la durata della simulazione, e un'istanza di ogni protocollo definito. Le relazioni dei nodi e le informazioni sul suo stato sono contenute nella `Social Table`, che definisce di fatto la sua ego network. La classe `Specific Node` implementa infine lo stato di ogni nodo, che può essere:

- `Online:` se il nodo è operativo nel sistema e si trova attualmente connesso;
- `Offline:` se il nodo è operativo nel sistema ma non è attualmente connesso;
- `Dead:` se il nodo non è più operativo nel sistema;
- `Down:` se il nodo non è `Dead`, ma risulta temporaneamente non operativo.

Nelle nostre simulazioni non è stato per il momento previsto il fallimento di alcun nodo, perciò i nodi si trovano sempre in uno degli stati `Online` o `Offline`.

4.4 Social Table

La Social Table è la componente che permette ad ogni nodo di accedere ai propri vicini e alle informazioni ad essi associate. Essa implementa l'interfaccia `Linkable` e mette a disposizione dei metodi per effettuare l'accesso alle informazioni in essa contenute. La Social Table è costituita da un insieme di entry di tipo $(Node, SocialTableEntry)$ che specificano, per ogni vicino, i valori della frequenza di contatto dall'ego a quello specifico

alter (`Out Contact Freq`), la frequenza di contatto dall'alter all'ego (`In Contact Freq`) e il valore della forza del legame (`Tie Strength`), ottenuta attraverso una combinazione delle precedenti due misure. La struttura della Social Table è mostrata in Figura 4.2, tratta da [15].

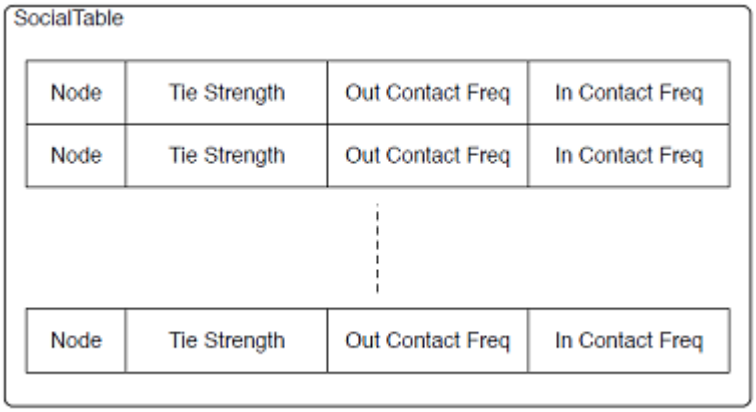


Figura 4.2: Struttura della Social Table

Sono presenti inoltre tre campi utili a gestire l'elezione delle social cache: il campo `EtichettaNodo` permette di sapere se il nodo è social cache o no. Esso, in accordo alla notazione utilizzata negli Algoritmi 5 e 6, assume il valore `BLACK` se il nodo è social cache e `WHITE` se non lo è. Abbiamo poi aggiunto per comodità il valore `RED` che indica che un nodo è una social cache che è anche il punto di memorizzazione locale per i dati di qualche nodo. I campi `FaseElez` e `FaseCop` contengono invece rispettivamente lo stato del protocollo di elezione dei punti di memorizzazione locali e del protocollo di copertura in esecuzione sul nodo. Il loro valore è stato spiegato nel Paragrafo 4.2.

Le entry della `Social Table` vengono inizializzate dalla classe `WireFromDatasetEgo` in base a quanto contenuto nel dataset configurato come parametro e vengono successivamente aggiornate grazie a un insieme di metodi che permettono di gestire al meglio le strutture dati. Il dataset è in formato *comma-separated values* (CSV) e contiene per ogni riga i nodi tra cui esiste la relazione e la frequenza di contatto dal primo nodo verso il secondo.

Il controller `NetworkObserver` permette di osservare l'overlay network accedendo alle Social Table dei nodi e leggendone il contenuto.

4.5 Modello di Churn

Il modello di Churn viene implementato attraverso un protocollo cycle-based contenuto nella classe `ChurnModel`. L'istanza del protocollo associata a ciascun nodo viene invocata periodicamente. Ogni istanza del protocollo contiene un contatore di tempo associato alla durata della sessione corrente che viene decrementato ad ogni ciclo. Quando il contatore arriva a 0 il nodo cambia stato (cioè passa da `Online` a `Offline` o viceversa) e il contatore viene resettato con il nuovo valore della durata della sessione estratto casualmente secondo il modello che verrà descritto nel seguito. Ad ogni connessione o disconnessione un nodo notifica ai propri vicini online il proprio cambiamento di stato, in modo che possano aggiornare opportunamente le rispettive strutture dati delle loro Social Table. Un nodo che torna online controlla inoltre che le proprie informazioni sui vicini siano aggiornate, dato che nel periodo in cui si trovava offline il suo stato è rimasto ovviamente inalterato. Deve infine verificare che tutti i suoi archi uscenti siano coperti ($outStar = 0$) e se così non è deve far ripartire l'algoritmo di copertura. Quando un nodo passa offline invece deve notificare ai propri vicini online la possibilità che essi abbiano a breve un arco scoperto: essi controllano il proprio valore dell'*outStar* calcolato considerando offline il nodo in fase di disconnessione e se esso è maggiore di 0 fanno ripartire l'algoritmo di elezione. Se il loro PML è uguale al nodo in fase di disconnessione devono far ripartire anche l'algoritmo di elezione. Infine se un nodo che passa offline è il PML per un vicino v che è già offline deve attivare la procedura di elezione per conto di v di un nuovo PML: ciò verrà fatto settando `FaseElez` a 2.

Descriviamo ora le distribuzioni utilizzate per generare i periodi di tempo in cui un nodo si trova online o offline.

Gli utenti di una OSN, come risultato anche dagli studi da noi condotti e riportati nella prima parte del Capitolo 5, sono caratterizzati da una disponibilità eterogenea e una durata media delle sessioni molto variabile. La probabilità che un utente si trovi online in un determinato istante di tempo varia radicalmente da un utente all'altro: è possibile modellare tale fenomeno attraverso un processo stocastico. Lo *user churn*, cioè il processo di arrivo e di abbandono del sistema da parte dei nodi, è caratterizzato da:

- **Online Session Distribution:** determina la durata media dei periodi di tempo in cui l'utente è online all'interno del sistema;
- **Offline Session Distribution:** determina la durata media dei periodi di tempo in cui l'utente è offline all'interno del sistema;

Questi due elementi caratterizzano la *disponibilità* del nodo all'interno del sistema. Per definire la disponibilità dei nodi utilizziamo il modello definito in [49], lievemente modificato per adattare la durata delle sessioni offline ai dati sperimentali da noi raccolti. Il modello è ottenuto considerando caratteristiche comportamentali degli utenti in social network reali.

L'idea del modello è attribuire secondo una certa distribuzione di probabilità a ogni utente un valore medio delle sessioni eterogeneo. Ogni utente, che avrà quindi un proprio valore medio `AverageOnTime` per la durata media delle sessioni online e un proprio valore medio `AverageOffTime` per la durata media delle sessioni offline, estrarrà, ogni qual volta dovrà effettuare un cambiamento di stato (da `Online` a `Offline` o viceversa), un valore per la durata della sua prossima sessione (online o offline) con una distribuzione di probabilità avente come valore medio `AverageOnTime` o `AverageOffTime`.

In altre parole la probabilità che un utente i sia online a un determinato istante t può essere modellato con un *processo di renewal* $Z_i(t)$ definito dalla seguente equazione:

$$Z_i(t) = \begin{cases} 1 \text{ se } i \text{ è online al tempo } t \\ 0 \text{ altrimenti} \end{cases} \tag{4.1}$$

In tale processo i periodi in cui il nodo i si trova online e offline sono ricavati da variabili casuali X^i_{on} e X^i_{off} ognuna delle quali ha associate una distribuzione F^i_{on} per descrivere la durata dei periodi online e una distribuzione F^i_{off} per descrivere la durata dei periodi offline. Diversi studi sull'argomento hanno mostrato come per descrivere la durata delle sessioni degli utenti sia naturale utilizzare distribuzioni di tipo *heavy-tailed* o esponenziali. Per questo motivo gli autori di [49] utilizzano per descrivere i periodi online e offline distribuzioni di Pareto di tipo II (la cui funzione di ripartizione è mostrata in Equazione 4.2) o distribuzioni esponenziali (la cui funzione di ripartizione è mostrata in Equazione 4.3). Nelle formule λ, α e β rappresentano i parametri delle distribuzioni.

$$pareto[\alpha, \beta] = F_i(x) = 1 - (1 + x/\beta_i)^{-\alpha_i}, x > 0, \alpha_i > 1 \tag{4.2}$$

$$exp[\lambda] = F_i(x) = 1 - e^{-\lambda_i x} \tag{4.3}$$

Vediamo ora come sia possibile generare delle distribuzioni che rispecchino l'eterogeneità delle sessioni dei peer dei diversi utenti. Indichiamo con $l_i = E(X^i_{on})$ e $d_i = E(X^i_{off})$ la durata media dei periodi di online e offline del nodo i. Per fare in modo che ogni utente sia caratterizzato da sessioni di lunghezza diversa generiamo casualmente per ogni utente i i valori l_i e d_i attraverso l'utilizzo di due distribuzioni di

Pareto di tipo II rispettivamente con $\alpha = 3$ e $\beta = 1$ per l_i e $\alpha = 3$ e $\beta = 15$ per d_i. Dato che il valore medio di una distribuzione di Pareto di tipo II è dato da $\frac{\beta}{\alpha-1}$ questi valori dei parametri ci permettono di avere un tempo medio di online e offline pari rispettivamente a $E(l_i) = 0,5$ ore e $E(d_i) = 7,5$ ore, che ben si accordano con i valori sperimentali misurati. Osserviamo che grazie a questi valori un utente medio dovrebbe avere un numero di sessioni online medio al giorno pari a $\frac{24ore}{(0,5+7,5)ore} = 3$, che ben descrive i valori reali. Purtroppo però la presenza di pochi utenti con durata media della sessione molto bassa (e quindi un conseguente numero di sessioni molto alto) non ci permette di avere un valore medio delle sessioni giornaliere per utente pari a quello desiderato. Il modello utilizzato non permette infatti di modellare allo stesso tempo la durata media dei periodi online e offline dei nodi e il numero delle sessioni per utente.

Dopo aver selezionato una coppia (l_i, d_i) per ogni utente i, la durata di una singola sessione (online o offline) viene ottenuta ad ogni cambio di stato dell'utente sempre attraverso distribuzioni esponenziali o di Pareto di tipo II, ma stavolta definite in modo che la loro media sia uguale ai corrispettivi valori l_i e d_i. Nel paper citato vengono proposte tre coppie di distribuzioni (riportate nelle Equazioni 4.4-4.6) da poter utilizzare per generare le durate delle singole sessioni. Si noti che tutte queste distribuzioni hanno i valori medi desiderati.

$$F^i_{on} \sim pareto[3, 2 \cdot l_i] \qquad F^i_{off} \sim pareto[3, 2 \cdot d_i] \tag{4.4}$$

$$F^i_{on} \sim pareto[1.5, l_i/2] \qquad F^i_{off} \sim pareto[1.5, d_i/2] \tag{4.5}$$

$$F^i_{on} \sim exp[1/l_i] \qquad F^i_{off} \sim pareto[3, 2 \cdot d_i] \tag{4.6}$$

Le diverse coppie di distribuzioni presentano grafi con "code" più o meno lunghe.

Per implementare questo modello, all'inizio della simulazione ogni utente estrae i propri valori medi per la durata delle sessioni online e offline nella classe `InitChurnModel` utilizzando le distribuzioni discusse. Ogni qualvolta un nodo passa da online a offline o viceversa estrae la durata della sua prossima sessione utilizzando la coppia di distribuzioni scelta tra quelle delle Equazioni 4.4-4.6 e prendendo come valori di l_i e d_i i propri valori medi estratti in precedenza.

Infine, in alcuni dei nostri esperimenti, come suggerito in [43], abbiamo imposto che una piccola percentuale di nodi scelti casualmente (dal 5% al 10%) fosse sempre online nel sistema: questa scelta è giustificata dalla crescente presenza di utenti che si connettono

alle social network tramite dispositivi mobili e rimangono poi connessi durante tutto l'arco del giorno pur non essendo pienamente attivi nella rete. Tali peer costituiscono degli ottimi punti di memorizzazione all'interno del sistema perché rimanendo sempre online garantiscono che i dati da loro memorizzati siano sempre disponibili per tutti gli utenti che ne fanno richiesta.

4.6 File di configurazione

Il file di configurazione definisce due tipi di impostazioni: quelle globali della simulazione e quelle locali delle singole componenti `Protocol, Control, Linkage, Transport,` etc. Il file è composto da coppie nome-valore (secondo la rappresentazione *java.util.-Properties*), mentre le righe che iniziano con il carattere "#" costituiscono commenti e sono quindi ignorate.

Nel Listing 4.2 vengono descritte le proprietà globali necessarie alla configurazione della simulazione. Ad ogni componente della simulazione viene associato un nome univoco. La proprietà `seed` specifica il seme del generatore (pseudo) casuale del simulatore e viene utilizzata per replicare esperimenti basati su comportamenti casuali. Vengono inoltre definiti il numero massimo di nodi della rete (`network.size`) e il numero di cicli di cui è composta la simulazione (`simulation.endtime`). PeerSim definisce la classe GeneralNode che viene utilizzata di default per la rappresentazione dei nodi. Tuttavia è possibile specificare un'implementazione diversa dei nodi attraverso la proprietà `network.node`.

Listing 4.2: Parametri globali della simulazione

```
# PeerSim Simulation

#Size of the network
SIZE 1000

#Number of cycles
CYCLES 500

#Random Seed
random.seed 1234567890

network.size SIZE
```

```
simulation.endtime CYCLES

#Node implementation
network.node it.simulation.data.SpecificNode
```

4.6.1 Inizializzatori

Gli inizializzatori da utilizzare nella simulazione sono definiti attraverso la proprietà `include.init` (come mostrato nel Listing 4.3) e vengono eseguiti una sola volta, prima dell'inizio dei cicli di simulazione. La proprietà `protocol` indica il protocollo che deve essere inizializzato.

Nella nostra simulazione abbiamo introdotto due inizializzatori. `WireFromDatasetEgo` si occupa di inizializzare le Social Table dei nodi: la proprietà `file` indica il nome del file in cui è memorizzato il dataset da cui estrarre la rete. `InitChurnModel` si occupa invece di inizializzare il protocollo che implementa il modello di Churn.

Listing 4.3: Inizializzatori

```
#Initializers
include.init initLoadDataset initChurn

#initChurn: Initialize Churn Model Protocol
#initLoadDataset: Initialize network topology from CSV dataset

#Inizializzatore: carica rete dal dataset
init.initLoadDataset it.simulation.init.WireFromDatasetEgo
init.initLoadDataset.protocol lnk
#File dataset NodeAID,NodeBID,freqContatto
init.initLoadDataset.file approximatedWeaklyComponent.csv

#Inizializzatore per churnModel churn
init.initChurn it.simulation.init.InitChurnModel
#Protocollo da inizializzare
init.initChurn.protocol churn
```

4.6.2 Protocolli

I protocolli implementati nella simulazione sono definiti attraverso la proprietà `include.protocol`, come mostrato nel Listing 4.4. Un protocollo di tipo cycle-based implementa il metodo `nextCycle`, il quale viene eseguito periodicamente in base ai parametri `from, step, until` definiti nel file di configurazione. La proprietà `from` specifica il ciclo di simulazione dal quale ha inizio l'esecuzione del metodo `nextCycle`, la proprietà `step` definisce ogni quanti cicli eseguire il metodo `nextCycle` mentre la proprietà `until` specifica il ciclo di simulazione in cui termina l'esecuzione del protocollo.

Nella simulazione abbiamo introdotto un protocollo per il livello di linking, implementato attraverso le Social Table, a cui attribuiamo una capacità iniziale pari a 1.

Per il protocollo `ChurnModel` andiamo a definire il tipo di distribuzione da utilizzare per generare i periodi di online e offline attraverso la proprietà `type`. Le tre distribuzioni descritte nel Paragrafo 4.5 sono caratterizzate da valori diversi di `type`. Questo protocollo può accedere sia a quello di linking che a quello per il modello di Churn.

Il protocollo `Copertura` può accedere alle Social Table che implementano il livello di linking e il protocollo `Elezione` può accedere sia alle Social Table che al protocollo per il modello di Churn.

Listing 4.4: Protocolli

```
#Protocols
include.protocol lnk churn cop elez

#Protocollo per routing table
protocol.lnk it.simulation.table.SocialTable
protocol.lnk.capacity 1

#Protocollo per la modellazione del Churn
protocol.churn it.simulation.CDprotocol.ChurnModel
protocol.churn.linkable lnk
#Tipo di distribuzione da utilizzare
DISTR_TYPE1 1
# F(x) = pareto(3,2*averageONtime)
# G(x) = pareto(3,2*averageOFFtime)
DISTR_TYPE2 2
# F(x) = pareto(1.5,averageONtime/2)
```

```
# G(x) = pareto(1.5,averageOFFtime/2)
DISTR_TYPE3 3
# F(x) = exp(1/averageONtime)
# G(x) = pareto(3,2*averageOFFtime)
protocol.churn.type DISTR_TYPE1
protocol.churn.step 1

#Protocollo Copertura
protocol.cop it.simulation.CDprotocol.Copertura
protocol.cop.linkable lnk
protocol.cop.step 1

#Protocollo Elezione
protocol.elez it.simulation.CDprotocol.Elezione
protocol.elez.linkable lnk
protocol.elez.step 1
protocol.elez.churn churn
```

4.6.3 Controlli

I controlli implementati nella simulazione sono definiti attraverso la proprietà `include.control`, come mostrato nel Listing 4.5. Come i protocolli essi vengono eseguiti durante i cicli di simulazione in base a quanto definito dalle proprietà `from, step` e `until`. Ogni controllo ha accesso alle funzionalità del protocollo specificato dalla proprietà `protocol`.

I controlli disponibili all'interno della simulazione sono i seguenti:

- `ControlChurnEvolution:` si occupa della schedulazione periodica del protocollo di churn;
- `ControlCopertura:` si occupa della schedulazione periodica del protocollo di copertura;
- `ControlElezione:` si occupa della schedulazione periodica del protocollo di elezione dei punti di memorizzazione locali;
- `SocialTableObserver:` stampa a video il contenuto della Social Table di ogni nodo;

- `Shuffle:` permette di variare in modo pseudocasuale l'ordine con cui vengono visitati i nodi su cui eseguire i protocolli;
- `NetworkObserver:` salva la rete nel file specificato dalla proprietà `filename`;
- `NetworkSocialScoreObserver:` stampa le informazioni sul Social Score nel file specificato dalla proprietà `filename`;
- `NetworkSocialCacheObserver:` stampa le informazioni sul caching nel file specificato dalla proprietà `filename`;

Listing 4.5: Protocolli

```
#Controls
include.control shf churnEvolution copEvolution elezEvolution
socialScoreObserver printSocialCacheInfo printGraph

#shf: Controllo per introdurre casualita' nella simulazione
#churnEvolution: Controllo per chiamare il protocollo di churn
#copEvolution: Schedulazione periodica del protocollo Copertura
#elezEvolution: Schedulazione periodica del protocollo Elezione
#socialScoreObserver: Informazioni sul Social Score dei nodi
#printSocialCacheInfo: Informazioni sul caching
#printGraph: Controllo per stampare a video la rete caricata
#printSocial: Visualizza la Social Table di ogni nodo

#Controllo per introdurre casualita' nella simulazione
control.shf Shuffle
control.shf.step 1

#Controllo per esecuzione periodica del protocollo ChurnModel
control.churnEvolution
        it.simulation.control.ControlChurnEvolution
control.churnEvolution.protocol churn
#eseguo inizialmente l'elezione sull'intera rete
control.churnEvolution.from 3
control.churnEvolution.until CYCLES-3
control.churnEvolution.step 1
```

```
#Controllo per esecuzione periodica del protocollo Copertura
control.copEvolution it.simulation.control.ControlCopertura
control.copEvolution.protocol cop
control.copEvolution.step 1

#Controllo per esecuzione periodica del protocollo Elezione
control.elezEvolution it.simulation.control.ControlElezione
control.elezEvolution.protocol elez
control.elezEvolution.step 1

#Controllo per la stampa delle informazioni sul Social Score
control.socialScoreObserver
        it.simulation.control.NetworkSocialScoreObserver
control.socialScoreObserver.protocol elez
control.socialScoreObserver.filename socialScoreInfoSS3.csv
control.socialScoreObserver.from 0
control.socialScoreObserver.step 1
control.socialScoreObserver.until 1

#Controllo per la stampa delle informazioni sul caching
control.printSocialCacheInfo
       it.simulation.control.NetworkSocialCacheObserver
control.printSocialCacheInfo.protocol elez
control.printSocialCacheInfo.filename socialCacheInfoSS3.csv
control.printSocialCacheInfo.from 1
control.printSocialCacheInfo.step 1

#Controllo per stampa a video della rete
control.printGraph it.simulation.control.NetworkObserver
control.printGraph.protocol lnk
control.printGraph.filename grafoRete.csv
control.printGraph.from 0
control.printGraph.step 1
control.printGraph.until 1
```

```
#Controllo per stampare contenuto della Social Table dei nodi
control.printSocial it.simulation.control.SocialTableObserver
control.printSocial.protocol lnk
control.printSocial.step 2
control.printSocial.from 0
control.printSocial.until 2
```

Capitolo 5

Risultati sperimentali

In questo capitolo presenteremo i risultati ottenuti con i nostri esperimenti. Il capitolo è diviso in due sezioni nettamente distinte: nel Paragrafo 5.1 mostreremo il nostro studio sulle caratteristiche di disponibilità degli utenti di alcune OSNs, i cui risultati sono stati anticipati nel Paragrafo 3.5.3 per definire la nuova formula del Social Score. Il Paragrafo 5.2 è invece dedicato ai risultati ottenuti con l'implementazione in PeerSim degli algoritmi proposti nel Capitolo 3. Saranno valutati diversi aspetti di tali algoritmi, che vanno dalle caratteristiche dei nodi che abbiamo individuato come potenziali punti di memorizzazione, al dinamismo delle social cache, fino al numero di nodi necessari alla copertura della rete.

5.1 Analisi availability-pattern in OSNs

In questo paragrafo studieremo la presenza di availability-pattern nelle OSNs e come essi possono essere sfruttati per descrivere i periodi online e offline dei peer. I risultati trovati sono stati utilizzati nel Capitolo 3 per la proposta di un algoritmo di social caching che tenga conto della dinamicità dei peer della rete. Vogliamo in particolare analizzare la presenza di comportamenti periodici (su base giornaliera o settimanale) o caratteristici riscontrabili nei tempi di connessione e disconnessione dei peer.

Diversi lavori (si veda ad esempio [8] e [21]) hanno mostrato come il riconoscimento di pattern ricorrenti nei periodi online e offline dei peer possa produrre significativi miglioramenti nelle prestazioni di numerosi algoritmi che risolvono problemi ricorrenti nei sistemi P2P. Esempi di tali problemi sono la diffusione delle informazioni all'interno della rete o il mantenimento di un corretto numero di copie di un'informazione per garantire che questa sia disponibile in qualsiasi momento agli altri utenti (problema

della *replication*). Tuttavia la presenza di availability-pattern è stata sfruttata solo marginalmente per le OSNs: per la nostra conoscenza attuale la quasi totalità dei lavori che sfruttano l'*availability* dei peer riguardano sistemi di file-sharing [7] o sono studi sul comportamento complessivo dei nodi all'interno di una social network. Il nostro obiettivo è invece, oltre a verificare la presenza di pattern temporali sull'intera rete, analizzare in che misura il comportamento di un singolo peer possa essere predetto correttamente sulla base del suo comportamento passato.

Vorremmo poi capire se gli eventuali pattern trovati possano essere usati per migliorare le performance di diversi algoritmi per le OSNs o se esistano altri fattori che ci permettono di descrivere in maniera sufficientemente precisa il comportamento temporale di un peer. Per far ciò costruiremo innanzitutto dei predittori che tentano di prevedere la presenza online di un peer in un determinato momento sulla base del fatto che lo stesso peer fosse online o meno nello stesso intervallo di tempo nei giorni precedenti.

Dopo aver introdotto le caratteristiche generali dei dataset da noi studiati (Paragrafo 5.1.1) analizzeremo i pattern globali riscontrabili nella rete (Paragrafo 5.1.2). Introdurremo quindi i predittori (Paragrafo 5.1.4) che verranno usati, come spiegato nel Paragrafo 5.1.5, per ricercare pattern comportamentali nei periodi online dei singoli peer. Dopo aver analizzato i risultati ottenuti nei Paragrafi 5.1.6, 5.1.7 e 5.1.8, proveremo a modificare gli esperimenti fatti variando diversi parametri (Paragrafo 5.1.9). Tenteremo quindi di estrarre un ridotto numero di peer con caratteristiche di alta predicibilità nel Paragrafo 5.1.10. Studieremo infine le caratteristiche delle sessioni dei peer nel Paragrafo 5.1.11 e riassumeremo tutti i risultati ottenuti nel Paragrafo 5.1.12.

5.1.1 Caratteristiche generali dei dataset

Per la nostra analisi abbiamo sfruttato i dataset relativi alle reti sociali di Tagged, Netlog e MySpace utilizzati in [24][1]. Tali dataset contengono informazioni sulla presenza online di più di 1000 utenti per ciascuna OSN campionati ad intervalli di un minuto. Gli utenti che non si sono mai connessi alla rete sono stati eliminati dai dataset. A ciascun utente è associato un identificatore univoco che ci permette di tracciarne il comportamento in sessioni diverse. Gli utenti sono stati monitorati per un periodo di 49 giorni consecutivi, dal 15 Marzo al 2 Maggio 2009. In alcuni dataset sono presenti alcuni record relativi al 14 Marzo 2009, ma dato che tali informazioni non sono complete esse non sono state considerate nella nostra analisi. Si segnala infine che la necessità di utilizzare dataset diversi per l'analisi della connettività degli utenti e per lo studio delle interazioni sociali

[1] I dataset sono disponibili online all'indirizzo `http://netecon_group.tmit.bme.hu/source-codes/online-social-network-dataset`

Dataset	**Tagged**	**Netlog**	**MySpace**
Numero Utenti Online	1657	1500	1723
Durata Connessioni Media (min)	34,87	82,46	14,98
Durata Connessioni Std. Dev.	22,32	67,67	9,58
Tempo Giornaliero Online Medio (min)	77,31	68,28	26,37
Tempo Giornaliero Online Std. Dev.	95,86	136,26	51,32
Numero Login Giornaliero Medio	2,21	0,85	1,34
Numero Login Giornaliero Std. Dev.	2,69	2,68	2,07

Tabella 5.1: Statistiche del dataset

tra di essi è dovuto alla mancanza di dataset che integrino le due informazioni. Nella Tabella 5.1 sono riassunte alcune delle caratteristiche principali dei dataset studiati.

5.1.2 Pattern globali e numero di utenti online

Per prima cosa verifichiamo la presenza di pattern giornalieri sull'intera rete. Ci aspettiamo che il numero di utenti online abbia un comportamento ciclico su base giornaliera e corrispondente ai normali ritmi di vita che caratterizzano utenti appartenenti allo stesso fuso orario. Tutte le social network analizzate sono infatti caratterizzate da una diffusione concentrata in specifiche aree geografiche e ci aspettiamo quindi che gli utenti monitorati, che hanno relazioni di amicizia tra loro, risiedano in larga parte nella medesima nazione. I comportamenti attesi, come mostrato in Figura 5.1, sono ampiamente riscontrabili nel dataset relativo a Tagged. Sull'asse x viene mostrato un numero progressivo per le finestre temporali: queste hanno una dimensione di un'ora, quindi le prime 24 finestre temporali corrispondono al primo giorno, quelle da 25 a 48 al secondo, etc. Tale intervallo corrisponde esattamente al periodo T del grafico che otteniamo plottando sull'asse y il numero di utenti online almeno una volta in tale finestra temporale. Come ci aspettiamo il numero di utenti online comincia ad aumentare nelle prime ore del mattino, tocca il picco massimo nella fascia serale e decresce quindi progressivamente fino ai valori minimi raggiunti nelle ore notturne.

Anche il grafo relativo a Netlog (Figura 5.2) presenta caratteristiche simili, anche se in alcuni giorni i pattern sono meno regolari: sono presenti infatti alcuni picchi "anomali" difficilmente prevedibili, che potrebbero corrispondere a particolari eventi che hanno portato a connettersi un alto numero di utenti.

Infine in Figura 5.3 sono mostrati i tre grafici relativi ai dataset di gruppi di utenti statunitensi, britannici e giapponesi di MySpace. Innanzitutto si osserva come i picchi

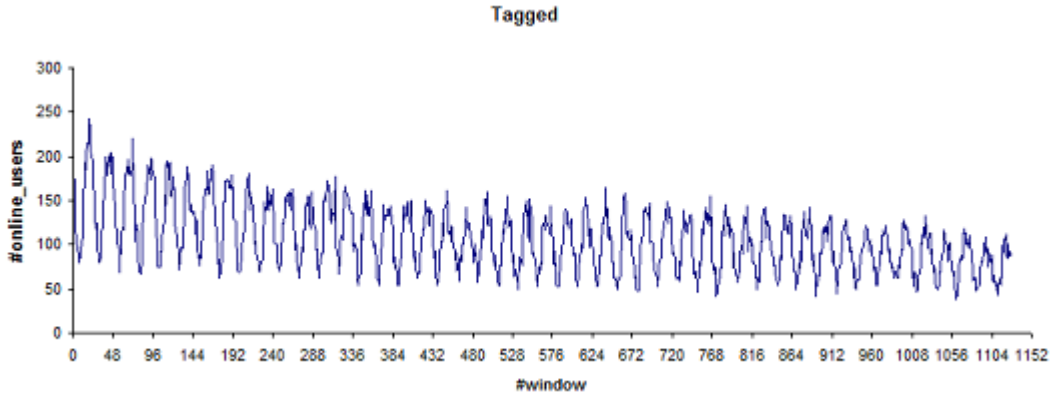


Figura 5.1: Periodicità del numero di utenti online in Tagged

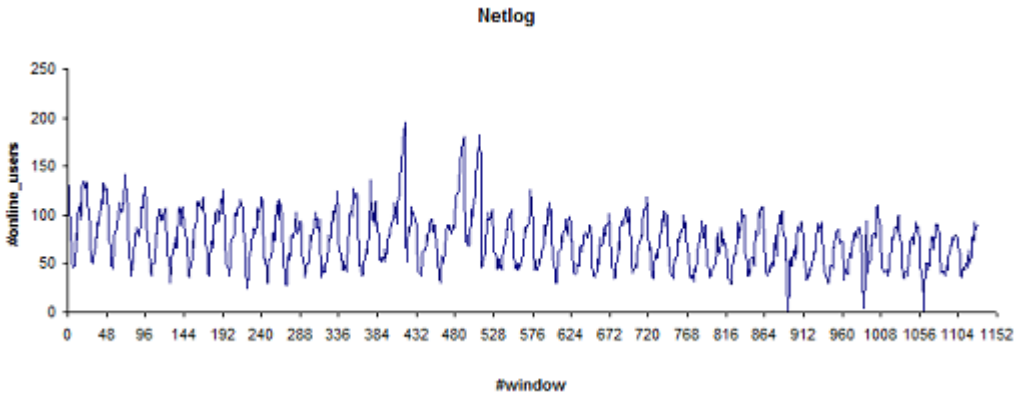


Figura 5.2: Periodicità del numero di utenti online in Netlog

e le altre regolarità del grafo siano traslate orizzontalmente: questo fenomeno è dovuto all'effetto del fuso orario esistente tra i diversi paesi; l'orario rispetto al quale sono state fatte le misurazioni è invece unico. Si osservi inoltre le peculiarità del dataset relativo agli utenti giapponesi: questo dataset contiene un numero di utenti inferiore agli altri (424, contro oltre 600 utenti degli altri due dataset) ma presenta, anche percentualmente, un numero molto inferiore di utenti online in ogni finestra temporale. Inoltre esso presenta caratteristiche di estrema regolarità, il che sembra far presumere un comportamento degli utenti piuttosto regolare.

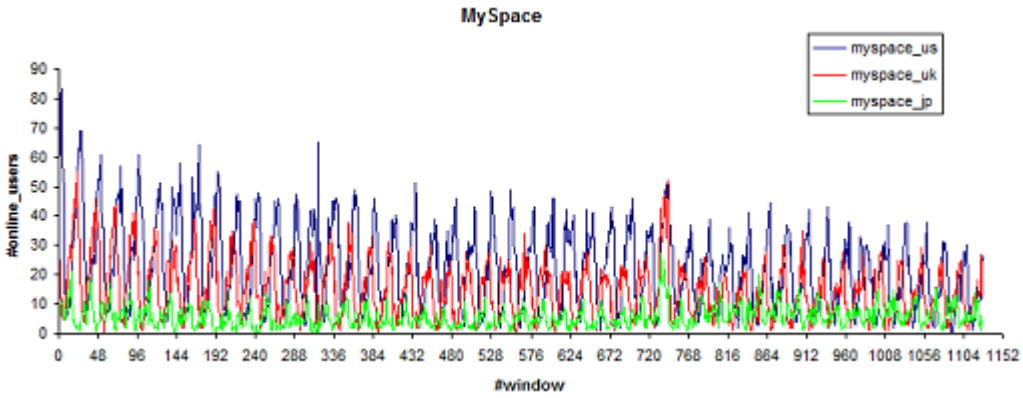


Figura 5.3: Periodicità del numero di utenti online in MySpace

In definitiva questa prima analisi sul numero degli utenti online nei diversi momenti della giornata ha mostrato come pattern temporali su base giornaliera siano evidentemente presenti in tutte e tre le OSNs analizzate.

5.1.3 Rappresentazione di una *temporal network* per il problema

Spostiamo adesso la nostra attenzione sulla ricerca di pattern nei periodi di connessione dei singoli peer. Più che alla presenza di ovvi pattern nei periodi offline, come ad esempio l'assenza di connessione della stragrande maggioranza dei peer negli orari notturni, vorremmo verificare la presenza di pattern nei periodi online. In particolare vorremmo analizzare quale percentuale dei peer ha comportamenti regolari, in modo da poterne così predire la presenza online con una certa accuratezza.

Per far questo creeremo dei predittori lineari che, osservando la storia passata dei peer, cercano di predire il loro comportamento futuro. Per il calcolo dei predittori dei tempi trascorsi online dai nodi siamo interessati a conoscere il comportamento tenuto

dal nodo nello stesso intervallo di tempo nei giorni passati. Per questo motivo occorre tenere traccia del grafo temporale dei periodi di connettività dei nodi. Per avere una misura compatta di tali periodi utilizzeremo una rappresentazione discretizzata con un intervallo di tempo Δt pari a 10min. Indichiamo con $G_d^n[i]$ il valore di connettività dell'$i-$esimo intervallo del giorno d per un nodo n: tale valore sarà pari a 1 se il nodo risulta online in almeno una delle 10 rilevazioni effettuate con frequenza di un minuto in tale intervallo, 0 altrimenti. In questo modo i periodi online di ciascun nodo in una giornata verranno rappresentati con un array di 144 bit.

La scelta di porre a 1 i bit del valore di connettività anche in presenza di un solo bit pari a 1 nell'intera finestra è una scelta importante: non vogliamo che i nostri predittori siano vincolati eccessivamente dalla presenza di un peer online esattamente nello stesso momento. Per evitare tuttavia che connessioni di durata eccessivamente breve ed estemporanee condizionino negativamente i nostri predittori è possibile rimuovere connessioni di durata inferiore ad una certa soglia: tuttavia questa operazione di raffinamento a priori dei dati non produce un significativo miglioramento delle prestazioni.

5.1.4 Realizzazione di un predittore

Vogliamo ora realizzare un predittore che stimi in modo efficace se un peer sarà online o meno in un determinato istante. Raffinando la proposta presente in [7] costruiamo un predittore lineare pesato che utilizza i *temporal graph* dei 7 giorni immediatamente precedenti a quello di cui si vuole predire la disponibilità del peer. Il nostro predittore per un generico nodo n avrà la forma:

$$Pr^n[i] = \frac{\sum_d w_d \cdot G_d^n[i]}{\sum_d w_d} \tag{5.1}$$

dove w_d sono dei pesi che indicano la diversa importanza attribuita alla presenza online dei peer nei diversi giorni e d sono i giorni che consideriamo nella predizione (i 7 giorni precedenti a quello attuale nel nostro caso).

Si osserva che $Pr^n[i] \in [0, 1]$ e in particolare i valori agli estremi dell'intervallo vengono assunti rispettivamente se il peer non è mai stato online o è stato sempre online nell'intervallo considerato nei 7 giorni precedenti. Osserviamo anche che il nostro predittore, essendo lineare, è estremamente semplice e può essere usato in un sistema P2P senza aumentare la complessità di calcolo in modo sensibile. È importante notare come non siamo interessati a costruire il miglior predittore possibile per stimare il futuro comportamento dei peer, ma vogliamo semplicemente analizzare in che misura il comportamento di un peer sia costante in giorni consecutivi. Esistono sicuramente predittori

più complessi, come quelli presentati in [37], in grado di produrre risultati migliori di quelli da noi ottenuti, ma certamente anche quelli da noi usati sono in grado di fornirci un'idea piuttosto chiara di quanto il comportamento di un singolo peer sia predicibile nel tempo.

Notiamo infine che con la nostra modellazione abbiamo ottenuto un predittore che ci fornisce una misura numerica della probabilità che un nodo sia online in un determinato intervallo temporale. Per i nostri esperimenti è tuttavia utile, anche al fine di ridurre la complessità di calcolo, avere un predittore binario che indichi semplicemente se si stima che il nodo sarà online o offline in un determinato intervallo temporale. Per fare ciò è sufficiente porre

$$Pr^n[i] = \begin{cases} 0 \text{ se } Pr^n[i] < \theta \\ 1 \text{ se } Pr^n[i] \geq \theta \end{cases} \tag{5.2}$$

dove θ è la soglia oltre la quale prediciamo che il nodo si troverà online. In [7] si assume che un peer sarà online se il predittore ha valori $\geq 5/7$, pari a circa 0,7. È evidente che, aumentando il valore della soglia, verranno predetti come online un numero sempre minore di nodi. D'altro canto la precisione della predizione, intesa come la percentuale di nodi effettivamente online tra quelli predetti online, crescerà progressivamente. I nostri esperimenti verranno eseguiti con valori diversi della soglia in modo da cercare un valore che rappresenti un giusto *trade-off* tra la precisione della predizione e il numero totale di finestre temporali in cui un peer viene previsto correttamente online.

Valore dei w_d nella formula dei predittori

Rimane da studiare con quale criterio attribuire il peso alla presenza online del peer nei giorni passati. Questo punto costituisce uno dei fattori che influenza maggiormente l'accuratezza del predittore e, naturalmente, si possono fare scelte differenti. Da risultati sperimentali (come quelli riportati in [6]) è emerso inoltre che diverse reti peer-to-peer hanno modelli di comportamento e frequenze di churn molto diverse e quindi i modelli di predizione possono variare sensibilmente.

Si possono però fare alcune osservazioni generali che ci possono essere di aiuto nella scelta dei pesi: innanzitutto le osservazioni relative ai giorni più recenti dovrebbero essere più significative di quelle dei giorni precedenti. Se indichiamo i giorni dal più recente al meno recente con gli interi da 1 a 7 una scelta ragionevole potrebbe essere ad esempio quella di porre $w_d = (7-d)+1$ in modo da assegnare ai pesi valori secondo una funzione lineare. In alcuni sistemi potremmo invece osservare un'alta correlazione tra la presenza

di un peer in un intervallo di tempo in due giorni consecutivi. Per accentuare l'importanza della presenza/assenza del peer nei giorni immediatamente precedenti potremmo alternativamente porre $w_d = 1/2^d$. In alcuni sistemi potrebbero emergere inoltre dei comportamenti con frequenza settimanale particolarmente rilevanti: ad esempio alcuni utenti potrebbero avere nel week-end un comportamento diverso da quello tenuto durante il resto della settimana. Per questo motivo in tali sistemi è utile aumentare il peso dell'osservazione relativa allo stesso giorno della settimana (cioè al giorno indicato con 7 secondo la nostra notazione). Infine in alcuni sistemi potremmo notare come i giorni della settimana precedente influenzino tutti nello stesso modo la presenza online di un peer in una determinata finestra temporale: in tal caso porremmo tutti i pesi pari a 1.

In generale la scelta dei pesi nella formula per il calcolo del predittore è un problema non banale che può variare sensibilmente da un sistema all'altro. Soltanto un'analisi accurata del sistema stesso e delle sue caratteristiche ci può guidare nella scelta di pesi che predicano in modo più accurato possibile il comportamento futuro del sistema.

5.1.5 Costruzione dell'esperimento

Vorremmo verificare sui nostri dataset l'efficacia dei quattro predittori ottenuti con le diverse scelte dei pesi discusse nel Paragrafo 5.1.4. In particolare analizzeremo i seguenti predittori:

- **predittore 1:** predittore con pesi lineari da 1 (giorno temporalmente più lontano) a 7 (giorno precedente a quello da predire);
- **predittore 2:** predittore con pesi lineari come il precedente ma peso pari a 7 anche per il giorno temporalmente più lontano: questo è infatti lo stesso giorno della settimana di quello di cui vogliamo fare la predizione e in questo modo accentuiamo l'importanza di eventuali pattern settimanali;
- **predittore 3:** predittore con pesi esponenziali da 1/128 (giorno temporalmente più lontano) a 1/2 (giorno precedente a quello da predire);
- **predittore 4:** predittori con pesi costanti tutti pari a 1.

Valuteremo i nostri predittori su tutte le finestre temporali a partire da quelle relative al nono giorno presente sul dataset (non possediamo le informazioni necessarie per calcolare correttamente il predittore per i giorni precedenti visto che anche i dati sul 15 Marzo sono solo parziali per alcune OSNs) fino al penultimo giorno (anche i dati sul 2 Maggio risultano essere in alcuni casi incompleti).

Calcoliamo il numero di finestre temporali in cui un peer previsto online si trova effettivamente online (true positive, TP), in cui un peer previsto offline si trova effettivamente offline (true negative, TN), in cui un peer previsto offline si trova invece online (false negative, FN) e in cui un peer previsto online si trova invece offline (false positive, FP). Per calcolare l'efficacia di un predittore utilizzeremo le seguenti misure:

- **accuratezza:** definita come $\frac{TP+TN}{TP+TN+FP+FN}$, che misura la percentuale di previsioni corrette;
- **attendibilità:** definita come $\frac{TP}{TP+FP}$, che misura la percentuale di previsioni corrette tra le sole finestre temporali in cui un peer è previsto online;
- **ampiezza predizione:** definita come $\frac{TP}{TP+FN}$, che misura la frazione di finestre temporali in cui il rispettivo peer si trova online che sono riuscito a prevedere correttamente con il mio predittore.

Tutti i predittori verranno testati con valori del parametro θ (la soglia oltre la quale un peer viene predetto online dell'equazione 5.2) che vanno da 0,1 a 1 con incrementi successivi di 0,1. Ci attendiamo, come accennato precedentemente, che l'attendibilità aumenti con θ e che l'ampiezza della predizione sia invece inversamente proporzionale rispetto ad essa.

Infine, con un piccolo abuso di notazione, definiremo finestre online (offline) le finestre temporali il cui rispettivo peer si trova online (offline): ciò ci permette di alleggerire notevolmente la scrittura.

5.1.6 Risultati per il dataset di Tagged

Come mostrano i risultati in Figura 5.4 l'accuratezza di tutti i predittori è decisamente buona: già con una soglia di 0,3 oltre il 90% delle previsioni sono esatte con tutti i predittori. Osserviamo però che le cose vanno molto peggio se andiamo ad analizzare i soli peer online o previsti online. Per ottenere un'attendibilità di previsione di oltre il 50% dobbiamo imporre una soglia di 0,6, andando di fatto ad individuare circa il 13% delle finestre online. Ciò è dovuto al fatto che gli alti valori dell'accuratezza sono principalmente dovuti alla corretta predizione dei periodi offline dei peer: questi sono la vasta maggioranza del totale (il che è in accordo col fatto che il tempo medio trascorso online dai nodi è di circa 77 minuti al giorno) e quindi portano a ottenere valori alti dell'accuratezza nonostante la difficoltà nell'individuare i periodi online. Tuttavia si osserva che anche un predittore che predicesse tutti i nodi sempre offline avrebbe ben il 94% di accuratezza!

Per questo motivo è necessario valutare soprattutto le altre due misure per capire la reale efficacia di un predittore. Notiamo che il predittore 3, quello con i pesi esponenziali, ci fornisce l'attendibilità decisamente peggiore. Questo implica che è scorretto sovrastimare eccessivamente l'importanza della finestra temporale riferita al giorno immediatamente precedente a quello da predire. Osserviamo che i predittori migliori sono il 2 e il 4: tuttavia, facendo ulteriori prove, il fatto che il predittore 2 fornisca migliori performance rispetto al predittore 1 non sembra da imputare alla presenza di significativi pattern settimanali quanto a una più omogenea distribuzione dei pesi, che più si avvicina a quella del predittore 4. Si osserva inoltre che il predittore 4 ha un andamento a scalini: questo è dovuto al semplice fatto che, non avendo pesi, esso può assumere solo i valori discreti da 0/7 a 7/7. Risulta quindi ovvio che il numero di previsioni corrette sia ad esempio lo stesso per valori della soglia pari a 0,6 e 0,7, dato che le finestre temporali con un valore del predittore maggiore di 0,6 avranno valori pari a 5/7, 6/7 o 7/7 e che tali valori sono tutti maggiori anche della soglia successiva ($5/7 \simeq 0,71 > 0,7$). Notiamo infine che per avere previsioni con alta attendibilità (>70%) individuiamo un numero molto ridotto di peer (inferiore al 5%).

In definitiva il comportamento degli utenti di Tagged risulta essere difficilmente predicibile: è possibile individuare con ottima efficacia un'alta percentuale di finestre offline (che sono un'ampia frazione del totale) ma i momenti in cui i peer sono connessi risultano poco prevedibili. Si osserva ad esempio che il 25% degli utenti che sono stati sempre online nella stessa finestra temporale nella settimana precedente, non lo saranno nel giorno della predizione. In altre parole se un peer risulta online per 7 giorni consecutivi in una data finestra temporale ho comunque una probabilità del 25% che esso non si trovi online nella stessa finestra temporale il giorno successivo. La percentuale di utenti di cui si riescono a prevedere i periodi online è in definitiva molto ridotta, perché ci sono un ampio numero di connessioni estemporanee non legate ad alcun pattern temporale particolare.

5.1.7 Risultati per il dataset di Netlog

Come mostrato in Figura 5.5 a prima vista i risultati generali ottenuti per il dataset di Netlog sembrano simili a quelli ottenuti per Tagged: l'accuratezza è buona e i predittori migliori sono il 2 e il 4. Le prestazioni tuttavia sono decisamente migliori. Riusciamo infatti a raggiungere valori dell'attendibilità più alti (fino all'85%), selezionando al contempo un numero maggiore di peer. Ad esempio per avere un attendibilità del 70%, riusciamo a individuare circa il 20% delle finestre online. Questo dimostra che il com-

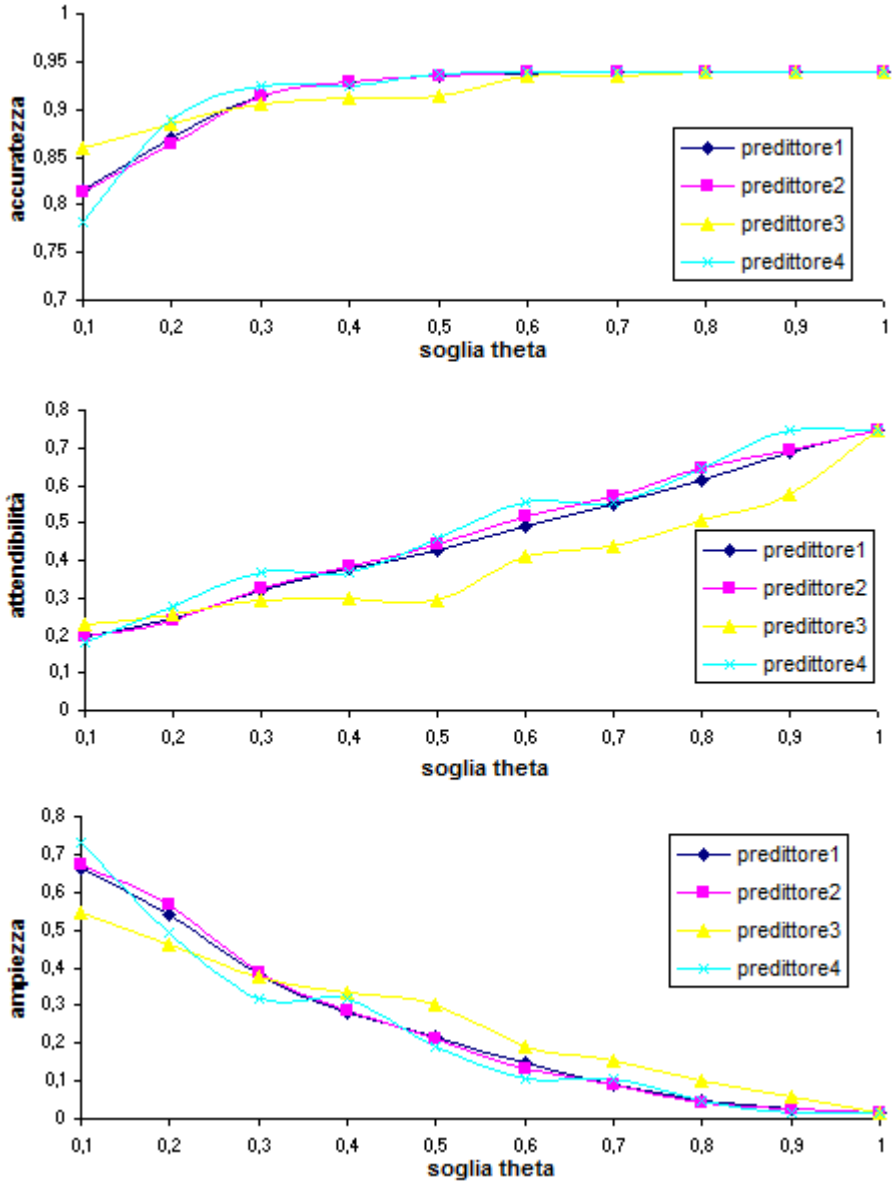


Figura 5.4: Accuratezza, attendibilità e ampiezza di predizione per il dataset di Tagged

portamento degli utenti di Netlog presenta una certa regolarità: dei peer online in un certo momento, una frazione consistente era online nella stessa finestra temporale in più giorni dell'ultima settimana. Andando inoltre ad analizzare il valore del predittore 4 per una soglia di 0,4 vediamo ad esempio che il 44% dei peer online in un dato momento era online nella stessa finestra temporale almeno 3 giorni su 7 nella settimana precedente.

Questi dati ci mostrano come in Netlog riusciamo a predire una frazione di finestre online assai più ampia rispetto a Tagged.

5.1.8 Risultati per il dataset di MySpace

Per MySpace abbiamo a disposizione 3 dataset, relativi rispettivamente a gruppi di utenti statunitensi, britannici e giapponesi. In Figura 5.5 mostriamo i risultati per i diversi dataset: riportiamo per semplicità di visualizzazione solo i grafici relativi al predittore 2 e al predittore 4 che, come per i dataset precedenti, si rivelano quelli che raggiungono prestazioni migliori. Notiamo che i tre dataset presentano caratteristiche piuttosto diverse: innanzitutto l'accuratezza è maggiore per gli utenti giapponesi rispetto agli altri due dataset ma, dopo un'analisi più approfondita, si comprende come questo sia dovuto al fatto che tali utenti hanno un tempo medio giornaliero online inferiore agli altri e che quindi ciò sia dovuto fondamentalmente alla corretta previsione di un più alto numero di finestre offline. Osserviamo però che anche l'attendibilità e l'ampiezza di predizione sono molto più elevate per il dataset degli utenti giapponesi rispetto agli altri due: per il dataset statunitense per raggiungere un 50% di attendibilità nella previsione degli utenti online riesco a individuare solo il 5% delle finestre online, per il dataset britannico circa l'8%, mentre per il dataset giapponese oltre il 20%. In modo simile per il dataset statunitense raggiungo al più il 71% di accuratezza; questo dato sale all'81% per il dataset britannico e addirittura al 91% per quello nipponico. In particolare su quest'ultimo dataset i predittori hanno prestazioni di poco inferiori a quelli del dataset di Netlog e riusciamo a predire con buona accuratezza una frazione non trascurabile delle finestre online (per avere il 70% di attendibilità individuiamo circa il 15% delle finestre online) mentre per gli altri due dataset i risultati sono decisamente peggiori.

Ciò ci porta dunque ad un'ulteriore conclusione: la predicibilità del comportamento degli utenti all'interno di una OSN, oltre che dalle caratteristiche della social network stessa, può variare da un gruppo di utenti all'altro. L'unica spiegazione plausibile alle osservazioni sopra riportate è infatti che gli utenti giapponesi abbiano un comportamento più regolare e metodico rispetto a quelli statunitensi e britannici e che ciò faciliti la previsione dei loro periodi online.

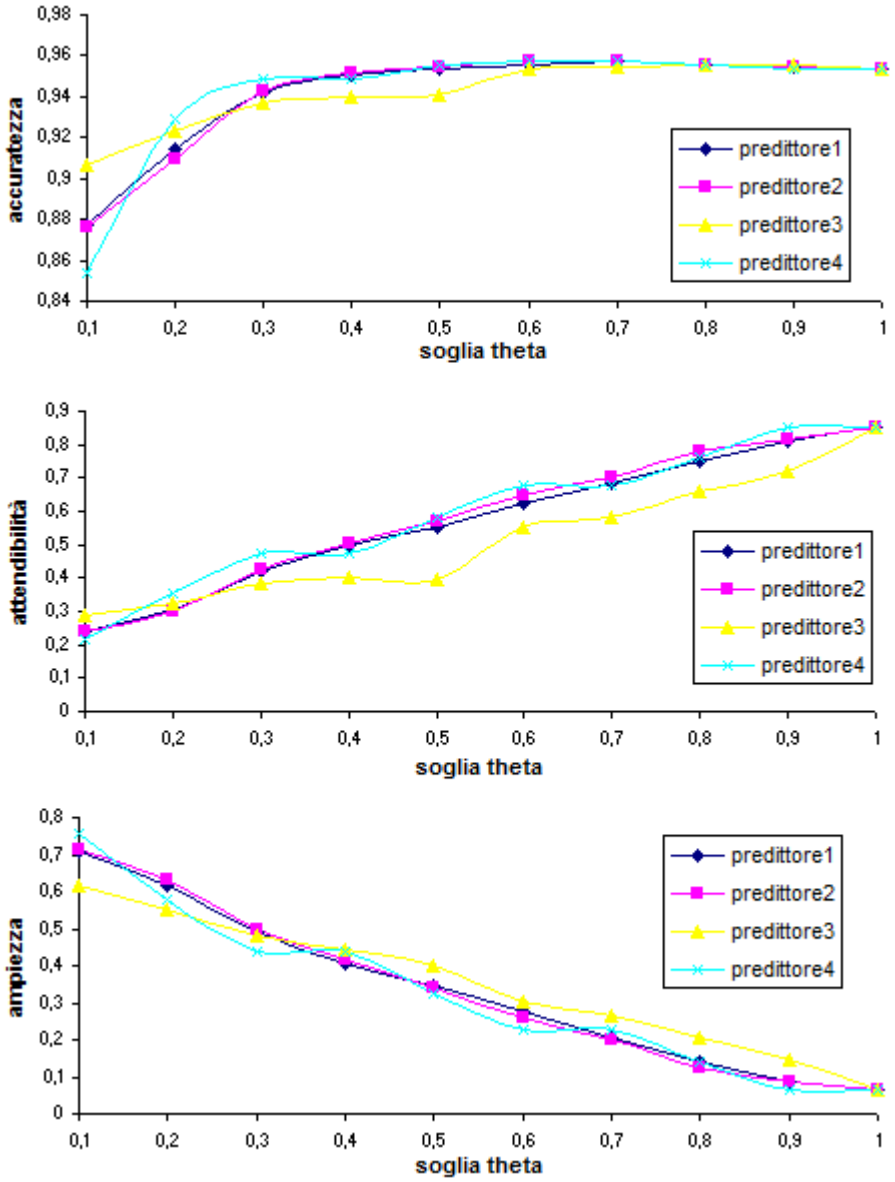


Figura 5.5: Accuratezza, attendibilità e ampiezza di predizione per il dataset di Netlog

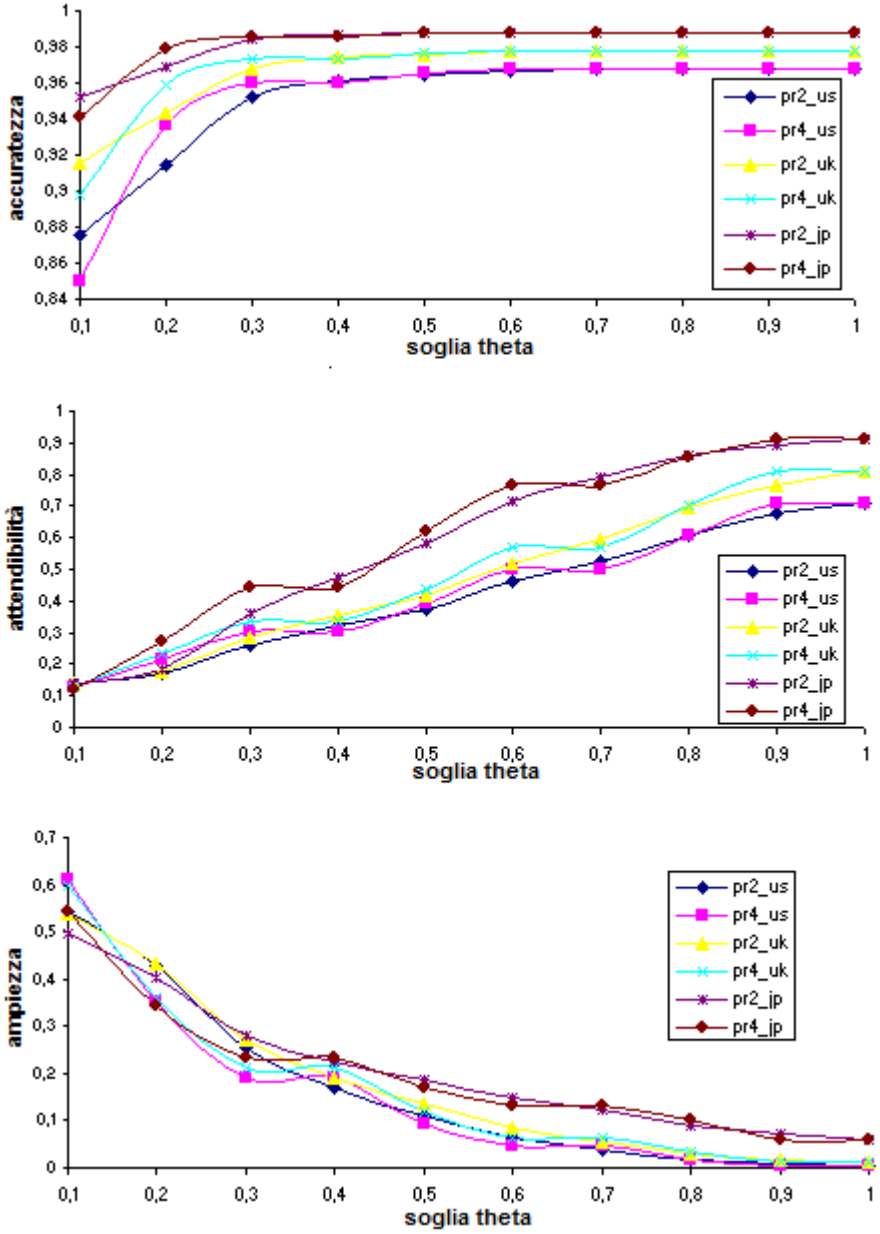


Figura 5.6: Accuratezza, attendibilità e ampiezza di predizione per i dataset di MySpace

5.1.9 Esperimenti con variazione di altri parametri

Facciamo ora alcune prove per vedere se riusciamo ad ottenere risultati significativamente migliori con la modifica di altri parametri.

Il primo tentativo è quello di aumentare la dimensione della finestra temporale da 10 a 20 minuti: in questo modo evitiamo di ricercare correlazioni tra periodi di tempo troppo ristretti e vediamo se esistono comportamenti temporali che si ripetono in lassi più ampi di tempo. Il miglioramento ottenuto nei predittori è generalmente compreso tra l'1% e il 2%, il che appare poco significativo in relazione anche alla perdita di precisione dell'analisi.

Un tentativo più interessante è quello di creare predittori lineari che considerino un più ampio intervallo temporale della storia passata dei peer per provare a prevedere il loro comportamento futuro. Creiamo perciò dei predittori simili a quelli studiati precedentemente ma che considerino i grafi temporali delle due settimane precedenti al giorno della previsione. In questo caso le variazioni nei risultati sono più evidenti: possedendo una visione più ampia del comportamento passato dei peer riusciamo a prevedere con più facilità il loro comportamento futuro. Il nostro obiettivo è quindi tentare di aumentare il numero di previsioni di finestre online corrette: questo accade, ma solo per valori piuttosto bassi dell'attendibilità. In Tagged riusciamo a passare dal 13% al 17% di ampiezza di predizione per i peer individuati con il 50% di sicurezza come online, ma solo dal 2,5% al 3,5% per i peer individuati con il 70% di sicurezza. L'unico vantaggio è, come atteso, che si riesce a raggiungere picchi più alti di attendibilità, al prezzo però di individuare pochissimi peer online: in Tagged si arriva all'86% di sicurezza contro il 75% precedente, ma vengono individuate solo lo 0,5% delle finestre online, il che appare piuttosto inutile. Per Netlog valgono considerazioni simili a quelle fatte per Tagged: in generale questo dataset, come osservato in precedenza, ha caratteristiche di predicibilità discrete ma, con la modifica apportata, le prestazioni migliorano di pochi punti percentuali. Tuttavia la presenza online del 3% dei peer può essere predetta con oltre il 91% di affidabilità. Infine i dataset di MySpace relativi agli utenti statunitensi e britannici presentano caratteristiche simili ai precedenti. L'unico dataset su cui i predittori forniscono performance visibilmente migliori è quello degli utenti giapponesi il quale, come osservato, presenta una buonissima regolarità. Si riescono infatti ad individuare il 16% delle finestre online con l'80% di attendibilità (contro il 12,5% precedente) e oltre il 10% delle finestre online con il 91% di attendibilità (contro il 6% precedente). Tuttavia tali risultati sembrano in parte dovuti a una regolarità del comportamento degli utenti giapponesi decisamente superiore alla media. In definitiva aumentando il periodo su cui

vengono costruiti i predittori otteniamo prestazioni lievemente migliori, senza tuttavia cambiare nella sostanza i risultati ottenuti con i predittori precedenti.

L'ultimo tentativo che facciamo per migliorare le performance dei predittori è quello di provare a eliminare a priori dal dataset le connessioni eccessivamente brevi (per esempio quelle di durata inferiore a 4 minuti) che potrebbero essere meno soggette a periodicità. Tuttavia le prestazioni risultano del tutto in linea con quelle riscontrate senza l'operazione di "pulitura" a priori dei dati e, anzi, per i dataset di MySpace, caratterizzati da connessioni di durata più breve, si assiste in alcuni casi a un lieve peggioramento delle prestazioni.

5.1.10 Estrazione di un insieme di peer predicibili

Il nostro prossimo tentativo è quello di provare ad estrarre da ciascun dataset un sottoinsieme di nodi con caratteristiche di alta predicibilità. Chiameremo tale sottoinsieme *predictable set*. Dato che il comportamento di un nodo risulta tanto più predicibile tanto più tempo esso trascorre online, risulta ragionevole supporre che se riuscissimo a estrarre un predictable set di dimensione anche ridotta, come fatto ad esempio in [7] per il problema della *replication*, avremmo informazioni temporali piuttosto precise su nodi che sono buoni candidati per assumere il ruolo di social cache.

Per tentare di ottenere risultati significativi lavoriamo innanzitutto sul dataset di Netlog, che risultava essere quello con caratteristiche di predicibilità più alte. Per tale dataset osserviamo che in media il 74% dei peer sta online meno di un'ora al giorno in ogni settimana: il comportamento di tali peer può essere approssimato come "sempre offline". Le loro connessioni non sono periodiche e non sono quindi predicibili con i metodi da noi usati. Un ulteriore 17% dei peer (in media), pur essendo online per più di un'ora al giorno, ha connessioni completamente aperiodiche dato che non si trova online in nessuna finestra temporale della settimana precedente in almeno 5 giorni su 7. Infine, mediamente, in ogni settimana esiste un 9% dei peer che sta online per oltre il 52% del tempo trascorso complessivamente online da tutti i peer. Tuttavia anche il comportamento di questi peer è predicibile solo parzialmente: per avere l'80% di attendibilità nella previsione delle finestre online ne individuiamo solo il 25%. Tale dato, che sarà certamente assai più basso per gli altri dataset, *mostra come la stragrande maggioranza delle connessioni dei peer in una OSN, a differenza di quanto avviene per i sistemi di file-sharing, sia impredicibile*: anche per i nodi parzialmente predicibili risultano periodiche solo una frazione ridotta delle connessioni, mentre le altre appaiono estemporanee.

5.1.11 Studio delle sessioni degli utenti

Nei paragrafi precedenti abbiamo osservato come il comportamento dei singoli peer sia difficilmente predicibile. Ciò che tuttavia fino a ora non abbiamo considerato è il fatto che, al momento dell'elezione di una social cache, possediamo un'ulteriore informazione che non veniva considerata dai predittori: i candidati ad essere eletti da un nodo n in un certo momento t sono solo i nodi della ego network di n online all'istante t. Inoltre il comportamento di un nodo è tanto migliore, tanto più esso rimarrà connesso nella rete. Quando esso passerà offline le informazioni da esso memorizzate dovranno infatti essere ridistribuite tra i peer rimasti online, generando un flusso di dati che deve essere quanto più possibile ridotto.

In [24] si osserva inoltre che il numero di connessioni giornaliere è in media decisamente basso mentre la durata media delle connessioni è, in relazione, piuttosto alta. Ciò ci fa sperare di riuscire a descrivere in modo piuttosto preciso il comportamento di un certo peer in base alle caratteristiche delle sue sessioni.

Analizziamo ora le caratteristiche delle sessioni degli utenti dei diversi dataset.

Osserviamo innanzitutto, come mostrato in Figura 5.7, che si riferisce al dataset di Tagged, che i peer sono caratterizzati da durata media delle sessioni piuttosto diverse. Questo risultato è valido per tutti e tre i dataset in esame. Dato che per buona parte dei peer la media viene calcolata su diverse decine (se non centinaia) di sessioni una differenza nella durata media delle sessioni anche di 10 minuti appare decisamente significativa.

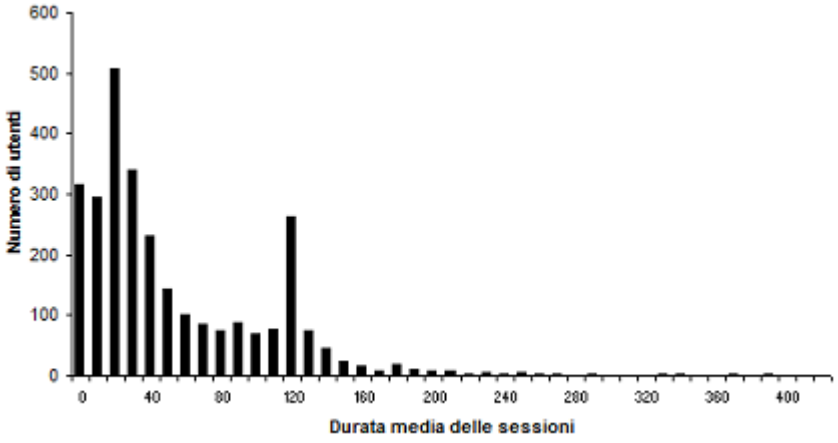


Figura 5.7: Durata media della sessione degli utenti in Tagged

Dagli esperimenti fatti possiamo osservare inoltre che un'alta percentuale dei peer è

caratterizzata da una durata media delle sessioni piuttosto lunga, mentre la deviazione standard delle stesse è in relazione piuttosto bassa (si veda Figura 5.8, che si riferisce ancora al dataset di Tagged).

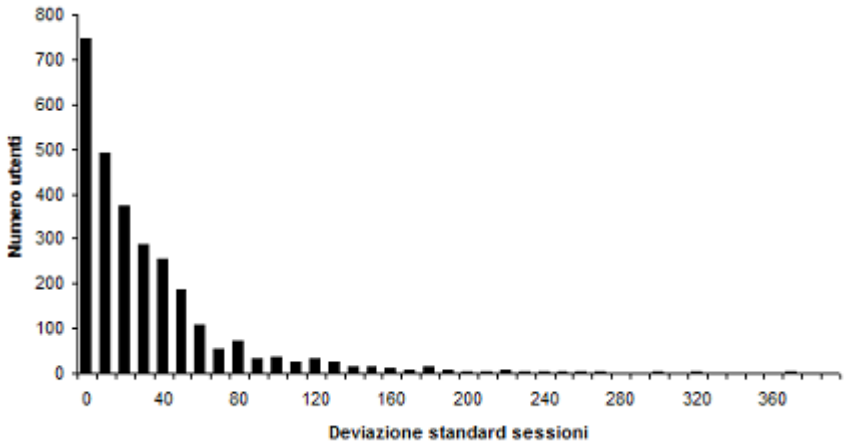


Figura 5.8: Deviazione standard delle sessioni degli utenti in Tagged

Si osserva inoltre un'ovvia correlazione tra la lunghezza media delle sessioni e la loro deviazione standard, ma questo è semplicemente dovuto al fatto che tanto più la durata media è alta, tanto maggiore è la possibilità che una sessione (per esempio più breve del normale) si discosti da tale valore medio. La correlazione tra durata media delle sessioni di un utente e la loro deviazione standard è mostrata in Figura 5.9.

Un interessante esperimento che abbiamo condotto sui diversi dataset è stato tentare di estrarre gruppi di utenti con caratteristiche delle sessioni simili attraverso algoritmi di clusterizzazione. Abbiamo cercato di capire se all'interno dei dataset ci fossero gruppi di utenti con caratteristiche delle sessioni simili tra loro e allo stesso tempo diverse da quelle degli altri gruppi di utenti. Purtroppo la nostra analisi ha prodotto risultati negativi: pur provando ad eseguire diversi algoritmi di clusterizzazione non siamo riusciti ad individuare cluster ben separati. La lunghezza della durata media della sessione degli utenti è distribuita infatti in maniera piuttosto omogenea e anche la loro deviazione standard non permette di caratterizzare i diversi gruppi in maniera significativa. I risultati ottenuti per il dataset di Netlog con l'algoritmo k-means e valore del parametro k pari a 7 è mostrato in Figura 5.10.

Esaminiamo infine alcune caratteristiche peculiari dei dataset delle diverse OSNs.

In Tagged il 73% dei nodi ha una durata media delle sessioni superiore a 20 minuti: ciò vuol dire che se troviamo un peer online che si è connesso abbastanza recentemente ci

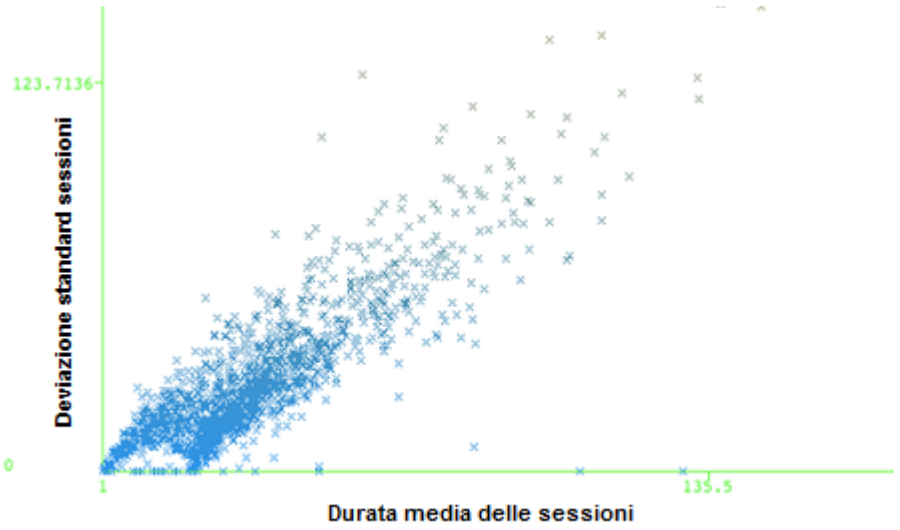


Figura 5.9: Correlazione tra durata media e deviazione standard delle sessioni degli utenti in Tagged

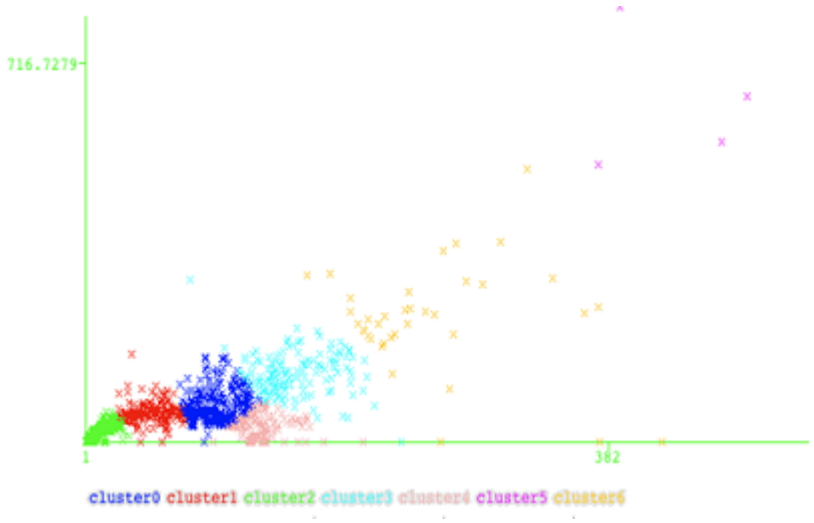


Figura 5.10: Cluster individuati da k-means sul dataset di Netlog con $k = 7$

sono ottime possibilità che esso rimanga online per un periodo piuttosto lungo di tempo. Il periodo che ci aspettiamo il nodo rimanga online è tanto maggiore tanto più alta è la durata media delle sue sessioni.

Netlog è una OSN caratterizzata da sessioni decisamente lunghe. In [24] viene infatti rilevato come la durata media della sessione degli utenti sia 82,46 minuti. Dalla nostra analisi osserviamo come addirittura il 40% dei peer abbia durata media delle connessioni superiore a 80 minuti e come, di tali peer, ce ne siano circa la metà per cui la deviazione standard delle sessioni è relativamente bassa. Questo indica che tutte le sessioni sono in media piuttosto lunghe.

MySpace è invece caratterizzato da sessioni di durata più breve (durata media riportata in [24] di 14,98 minuti), ma è comunque riscontrabile una buona eterogeneità tra le durate medie delle sessioni dei diversi utenti.

In definitiva, nonostante la durata delle sessioni dei peer non sia perfettamente uniforme, sembra possibile individuare nodi con durate medie decisamente diverse. Ci proponiamo di utilizzare quindi queste caratteristiche per stimare il tempo che un peer connesso trascorrerà online prima di disconnettersi.

5.1.12 Conclusioni

Dall'analisi fatta possiamo osservare, in pieno accordo a quanto affermato in [26], sezione II, come sia complesso predire il comportamento dei singoli utenti all'interno di una social network. Molti utenti hanno un comportamento difficilmente predicibile e, nonostante la presenza di availability-pattern globali sull'intera rete, la presenza di connessioni estemporanee e di eventi occasionali che possono portare alla connessione o alla disconnessione degli utenti comporta che sia di fatto impossibile predire i periodi online degli utenti con adeguata sicurezza. Uno dei fattori che rende difficile tale compito è inoltre la scarsa presenza online di molti utenti, che rende di fatto più difficile l'individuazione di pattern temporali. Questa è una caratteristica tipica delle OSNs, che le distingue nettamente dai sistemi di file-sharing. Le proposte che sfruttano gli availability-pattern per migliorare la distribuzione delle informazioni in tali sistemi riescono a predire in modo piuttosto preciso il comportamento di alcuni peer della rete grazie anche all'ampia frazione di tempo che essi trascorrono online: ciò non accade nelle OSNs.

Come atteso, abbiamo osservato come l'accuratezza e l'attendibilità dei predittori aumenti con la soglia θ, mentre l'ampiezza della previsione decresce all'aumentare di essa. I predittori 2 e 4 si sono rivelati i migliori per tutti e tre i dataset, mentre il predittore 3 fornisce le prestazioni peggiori, sovrastimando l'importanza dell'osservazione relativa al

giorno immediatamente precedente a quello di cui si vuole fare la previsione. È risultato impossibile anche estrarre un sottoinsieme di utenti con comportamento completamente predicibile.

Abbiamo invece osservato come sia possibile descrivere il comportamento temporale di un peer attraverso le caratteristiche delle sue sessioni. All'interno delle diverse OSNs sono presenti peer caratterizzati da sessioni più o meno lunghe ma piuttosto omogenee per ogni singolo utente: la diversità della durata media delle sessioni dei singoli utenti può essere certamente utilizzata per stimare il tempo che un peer online trascorrerà nel sistema prima di disconnettersi.

5.2 Valutazione del modello proposto

In questa seconda parte del capitolo discuteremo alcuni dei risultati ottenuti tramite gli esperimenti condotti per la valutazione del nostro modello. In particolare la nostra attenzione si concentrerà sulla valutazione degli algoritmi di elezione e rielezione dei punti di memorizzazione locali proposti e sulle caratteristiche dei punti di memorizzazione individuati. Nel Paragrafo 5.2.1 descriveremo le caratteristiche principali del dataset utilizzato per i nostri esperimenti. Discuteremo poi alcune caratteristiche della rete su cui condurremo i nostri test: ci occuperemo innanzitutto dell'algoritmo utilizzato per estrarre dall'intero dataset reti di dimensioni minori che possono essere gestite da PeerSim. La nostra attenzione si concentrerà quindi sulla valutazione delle sessioni degli utenti che otteniamo con il modello di Churn definito nel Paragrafo 4.5 e al loro confronto con le sessioni di utenti reali. Analizzeremo quali delle caratteristiche delle reti estratte sono simili a quelle delle reti reali e quali invece vanno perse. Il Paragrafo 5.2.3 è dedicato alla valutazione delle caratteristiche dei punti di memorizzazione individuati con il nostro algoritmo di elezione: esamineremo il loro punteggio di Social Score confrontandolo con quello medio dei nodi della rete e discuteremo le loro caratteristiche di disponibilità e di centralità. Verrà inoltre individuato il livello di persistenza nella rete che i punti di memorizzazione offrono con il soddisfacimento dei vincoli di *trust* e indagheremo infine il grado di copertura garantito dai punti di memorizzazione in seguito alla disconnessione di alcuni nodi. Infine nel Paragrafo 5.2.4 misureremo il numero totale di social cache necessarie per garantire la copertura completa di reti di diverse topologie.

5.2.1 Caratteristiche del dataset

Il dataset considerato è ottenuto da una Facebook Regional Network[2] contenente, oltre a un grafo non orientato che descrive l'intera struttura della rete, quattro *Interaction Graph*, grafi non orientati che descrivono le interazioni tra gli utenti in quattro diverse finestre temporali, pari a un mese, 6 mesi, un anno e tre anni e mezzo. Le osservazioni coprono un periodo che va dal 2004 al 2008. Nel dataset non è contenuta alcuna informazione riguardo al momento in cui si è instaurato il legame di amicizia tra gli utenti, che è quindi approssimato attraverso la prima interazione avvenuta tra loro. Il grafo completo contiene 3 097 165 nodi e 23 667 394 archi. Non è nota la tecnica con cui sono stati selezionati gli utenti della rete.

Si è eseguito poi un processo di raffinazione che ha portato all'estrazione di un dataset più adatto ai nostri scopi. Sono state eliminate le relazioni di amicizia a cui non corrisponde alcuna interazione e per ognuna delle relazioni attive sono stati prodotti due archi pesati con la frequenza di contatto tra i due utenti, ottenuta considerando il numero di interazioni avvenute tra i due peer nella finestra temporale. Dal dataset sono stati quindi rimossi tutti gli utenti che hanno avuto la loro prima interazione negli ultimi 6 mesi e quelli che hanno, in media, meno di 10 interazioni per mese. Il dataset risultante contiene i dati sui contatti di 86 635 nodi: per ogni interazione orientata tra una coppia di nodi A e B, il dataset contiene una riga $(idA, idB, freq_{AB})$ dove idA è l'identificatore unico del nodo A, idB è l'identificatore unico del nodo B, e $freq_{AB}$ è la frequenza di contatto dal nodo A verso il nodo B.

L'incapacità di PeerSim nel trattare contemporaneamente un numero così elevato di nodi ha portato alla necessità di introdurre algoritmi per estrarre sottoreti di dimensioni più ridotte, che potessero essere gestite più agevolmente da PeerSim.

5.2.2 Caratteristiche generali della rete

In questo paragrafo ci occuperemo delle caratteristiche generali del nostro modello. Analizzeremo innanzitutto le distribuzioni dei gradi dei nodi ottenute con l'algoritmo di estrazione e, successivamente, studieremo le caratteristiche delle sessioni degli utenti ottenute con il nostro modello di Churn.

[2]Disponibile all'indirizzo `http://current.cs.uscb.edu/facebook/`

Algoritmo di estrazione e grado dei nodi

L'estrazione di una rete di dimensioni ridotte (tale che possa essere gestita da PeerSim) da un dataset di dimensioni molto maggiori è un problema non irrilevante e ha un profondo impatto sui risultati degli esperimenti che condurremo. Purtroppo è impossibile riprodurre tutte le caratteristiche di una rete di grandi dimensioni in una rete più piccola e ciascuna delle diverse reti che è possibile estrarre riesce a descrivere meglio alcuni aspetti della rete completa ma peggio altri.

L'algoritmo per l'estrazione di una rete di dimensioni ridotte proposto in [15] sembra estrarre una rete con una topologia molto particolare, dominata da nodi con gradi molto bassi, che rispecchia poco fedelmente le caratteristiche di una rete reale. Abbiamo quindi proposto un nuovo algoritmo di estrazione, riportato in Algorithm 11, che estrae una rete con una distribuzione dei nodi completamente diversa.

L'algoritmo ripete ciclicamente, fino al raggiungimento della dimensione del dataset voluta, la seguente procedura: ordina i nodi del dataset in base al grado e elimina la metà dei nodi con grado più basso. Elimina quindi dai contatti dei nodi "sopravvissuti" i nodi scartati al passaggio precedente e riordina l'array in base al nuovo grado dei nodi. Al termine della procedura vengono introdotti tutti gli archi esistenti tra i nodi inseriti nel grafo: i nodi scelti per l'inserimento sono caratterizzati da grado medio molto alto.

Algorithm 11 Estrazione casuale della rete dal dataset

```
function LOAD(dataset,maxNetworkSize)
    V = dataset;                                  ▷ insieme dei nodi ancora non eliminati
    while V.size > maxNetworkSize do
        V = orderByDegree(V);                     ▷ ordinamento nodi per grado decrescente
        removeElementsInRange(max(V.size/2, maxNetworkSixe), V.size);
        refreshNodeDegree(V);                                  ▷ ricalcola grado nodi
    end while
    E = getIncidentEdges(V, dataset);             ▷ archi incidenti ai nodi selezionati
     return G = (V, E);
end function
```

La rete ottenuta, la cui distribuzione dei gradi è mostrata in Figura 5.11, ha un grado medio pari a 47,3 e una distribuzione dei gradi molto più bilanciata rispetto a quella dei nodi della rete ottenuta con l'algoritmo proposto in [15], con andamento gaussiano e una lunga coda di nodi con grado molto maggiore della media. Questa rete sembra essere un modello molto più realistico per i nostri esperimenti. Ci sono comunque aspetti di una rete reale che non riusciamo a evidenziare con tale rete: tra queste ci sono la presenza di un piccolo numero di peer caratterizzati da grado basso e la presenza di cluster all'interno

della rete. Quest'ultima caratteristica si sta accentuando negli ultimi anni di evoluzione delle social network ed è quindi sottorappresentata anche nell'intero dataset, che risale al 2008.

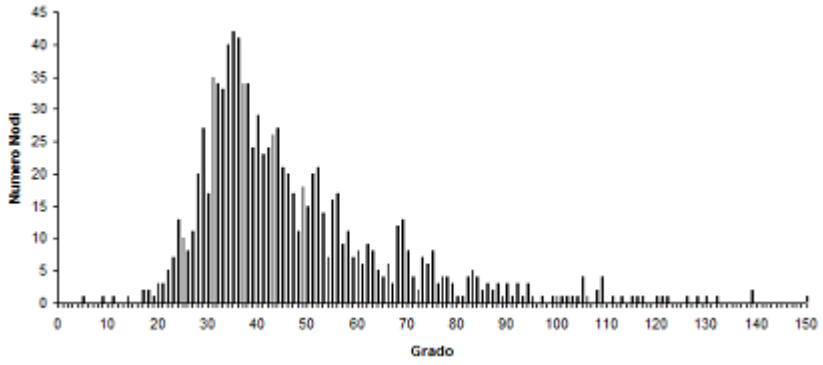


Figura 5.11: Distribuzione gradi in una rete di 1000 nodi estratta con Algorithm 11

Valutazione delle sessioni ottenute con il modello di Churn

In Figura 5.12 viene riportata la distribuzione della durata media delle sessioni giornaliere di 1000 nodi ottenute utilizzando il modello di Churn implementato.

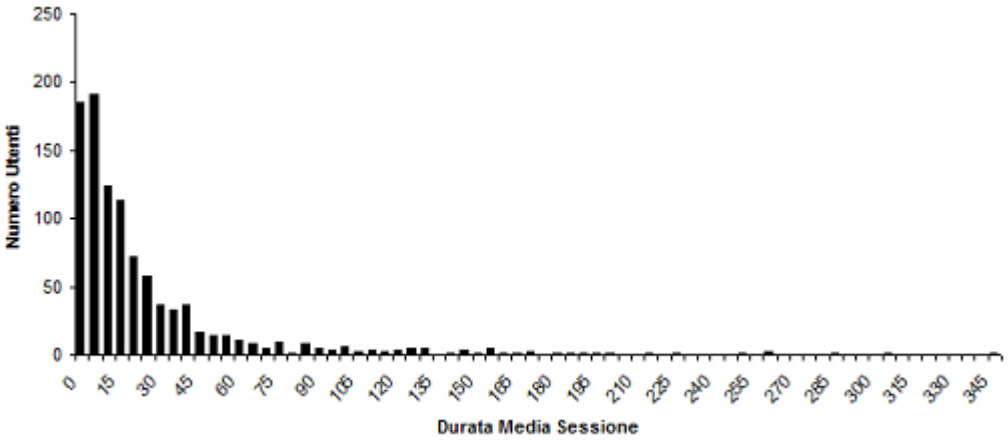


Figura 5.12: Distribuzione della durata delle sessioni degli utenti estratte con il modello di Churn

Osserviamo che la distribuzione, come atteso, è di tipo *heavy-tailed*: la media della durata delle sessioni è di circa 28 minuti (in ottimo accordo con il valore teorico atteso di 30 minuti) ma è presente una lunga coda di nodi con durata della sessione molto maggiore della media. Come discusso nel Paragrafo 4.5 purtroppo il numero di sessioni giornaliere per utente è decisamente superiore al valore reale: ci sono infatti molti utenti caratterizzati da sessioni online e periodi offline entrambi molto brevi che, a causa della loro alternanza costante, hanno un numero di sessioni giornaliere elevatissimo. In assoluto comunque il modello riesce a catturare l'eterogeneità della durata media della sessione degli utenti, che è di fatto l'aspetto di nostro interesse.

5.2.3 Punti di Memorizzazione

In questo paragrafo vogliamo studiare le caratteristiche dei nodi selezionati come punti di memorizzazione con il nostro algoritmo. Un alto numero degli esperimenti descritti nel seguito sono stati condotti su una rete di 1000 nodi estratta con Algorithm 11: tale scenario verrà nel seguito chiamato *Scenario Standard*.

Social Score

Lo scopo del primo esperimento è quello di verificare che il nostro algoritmo di elezione individui nodi caratterizzati da un alto punteggio di Social Score.

A questo scopo misuriamo nello *Scenario Standard* il valore medio del Social Score nei primi 100 cicli di simulazione di tutti i nodi e dei soli nodi che sono il punto di memorizzazione locale per i dati di almeno un vicino. All'inizio della simulazione la rete è in una configurazione in cui tutti i peer sono connessi, successivamente i nodi iniziano a disconnettersi e riconnettersi alla social network in base al modello di Churn. Nella finestra temporale considerata si osserva quindi una progressiva diminuzione del numero di nodi nel sistema fino al ciclo 40 dopo di che il numero dei nodi connessi rimane piuttosto stabile. I risultati misurati sono riportati in Figura 5.13.

Inizialmente il punteggio di Social Score dei Punti di Memorizzazione Locali (PML) è nettamente più alto di quello degli altri peer: i PML hanno infatti un punteggio medio di circa 70 000, circa 7 volte maggiore di quello di un normale nodo. Questo indica come i PML costituiscano degli ottimi punti di memorizzazione per la rete. Progressivamente la differenza si assottiglia: i PML tendono a rimanere nella rete, ma il loro punteggio di Social Score diminuisce perché molte triadi transitive, a causa della disconnessione progressiva degli altri nodi, non sono più presenti nella rete. Ricordiamo infatti che un arco tra due nodi offline non è considerato nel calcolo delle triadi transitive. Il punteggio

di Social Score medio dei nodi decresce invece più lentamente: se da un lato è infatti vero che l'osservazione fatta sulla scomparsa delle triadi transitive è valida per tutti i nodi, è dall'altro lato vero anche che tendono a rimanere nella rete i nodi con punteggio di Social Score maggiore, che sono caratterizzati da un *EstimatedUptime* più alto: tali nodi fanno aumentare il punteggio di Social Score medio. L'effetto combinato di questi due fattori rende la variazione del punteggio di Social Score medio dei nodi meno ampia di quella dei soli PML.

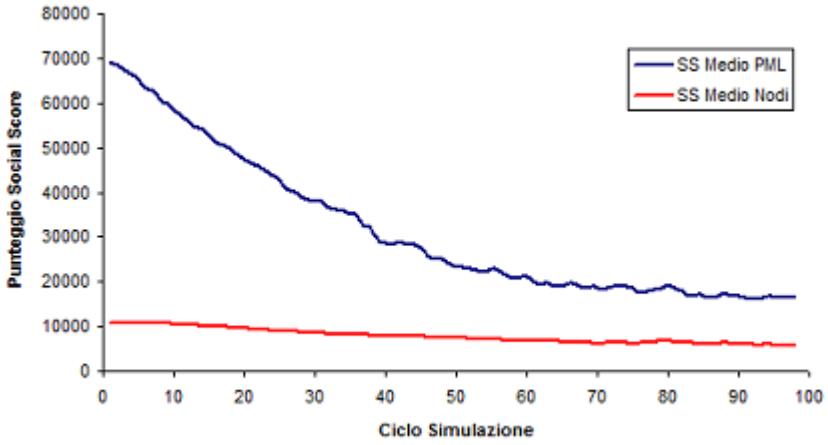


Figura 5.13: Evoluzione punteggio Social Score di tutti i nodi e dei PML

Comportamento temporale

Uno degli obiettivi di questa tesi è quello di riuscire a individuare *punti di memorizzazione il più possibile stabili nel tempo*. In particolare, vogliamo verificare che grazie all'introduzione del termine *EstimatedUptime* nella formula per il calcolo del Social Score si riesce a individuare nodi che rimarranno a lungo nella rete.

Lo scopo di questo esperimento è verificare che l'*EstimatedUptime* dei nodi che sono stati scelti come punto di memorizzazione locale da almeno un vicino è nettamente superiore rispetto al valore medio dell'*EstimatedUptime*. In Figura 5.14 viene riportato l'andamento del valore medio dell'*EstimatedUptime* per i PML e per tutti i nodi nei primi 100 cicli di simulazione nello *Scenario Standard*.

Inizialmente i nodi sono tutti online e quindi il loro *EstimatedUptime* medio è pari alla durata media della sessione (circa 30 minuti). Il valore dell'*EstimatedUptime* medio per i PML è inizialmente circa 4 volte maggiore. Nei cicli successivi alcuni nodi iniziano

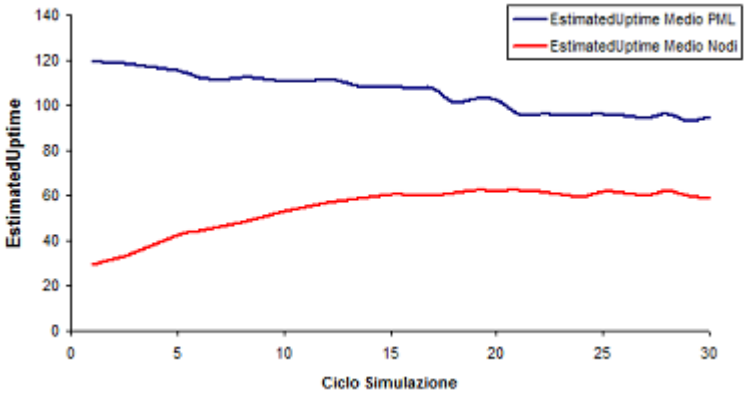


Figura 5.14: Evoluzione *EstimatedUptime* di tutti i nodi e dei PML

ad abbandonare la rete: questi sono nodi caratterizzati da durata media della sessione bassa e quindi vorremmo che non fossero i nodi da noi selezionati come punti di memorizzazione. Osserviamo che il valore dell'*EstimatedUptime* medio dei PML diminuisce leggermente, mentre quello complessivo dei nodi aumenta. Se andiamo ad esaminare più attentamente i dati notiamo che inizialmente l'*EstimatedUptime* dei nodi diminuisce di circa un'unità a ogni ciclo: un ciclo di simulazione corrisponde per noi a un minuto e l'*EstimatedUptime* dei PML che rimangono online diminuisce di un'unità per ogni minuto passato offline: ciò è una prova del fatto che nei primi cicli di simulazione i nodi selezionati come PML tendono a rimanere tutti online. L'*EstimatedUptime* dei nodi generici invece aumenta: l'*EstimatedUptime* dei nodi che rimangono online diminuisce di un'unità a ogni ciclo ma, d'altra parte, i nodi caratterizzati da un *EstimatedUptime* basso sono quelli che hanno maggior probabilità di abbandonare il sistema. L'effetto combinato di questi due fenomeni produce quindi un aumento dell'*EstimatedUptime* dei nodi generici. La scelta di PML con *EstimatedUptime* alto produce anche una drastica riduzione del numero di rielezioni dei punti di memorizzazione.

Persistenza dell'informazione nella rete

Nei Paragrafi 3.6 e 3.7 abbiamo proposto un insieme di algoritmi per garantire la persistenza dei dati di ogni peer nella rete tramite l'elezione di un Punto di Memorizzazione Locale per i dati di ciascun nodo. Abbiamo però discusso di un possibile problema che si può verificare: se un PML che si sta disconnettendo dal sistema non ha alcun vicino comune online con uno dei peer p offline di cui è PML, esso non può garantire al contem-

po la persistenza dei dati di p nella rete e il rispetto dei vincoli di *trust*. Abbiamo però affermato che tale scenario si verifica raramente, dato che i PML vengono scelti in base alla loro centralità nella rete. Scopo di questo algoritmo è verificare in che misura la nostra strategia garantisca la persistenza dei dati di tutti i peer nella rete senza ricorrere al passaggio o alla memorizzazione dei dati di un utente su utenti che non sono collegati ad esso da un vincolo di amicizia.

In una simulazione eseguita nello *Scenario Standard*, quando almeno il 38% dei nodi della rete era online era garantita la persistenza dei dati di tutti gli utenti, cioè ciascun nodo n aveva in ogni momento un proprio punto di memorizzazione locale PML_n selezionato tra i nodi facenti parte della sua ego network. Quando i nodi online scendevano al 30% del totale, non era più possibile garantire la persistenza dei dati di circa il 2% degli utenti con il contemporaneo completo soddisfacimento dei vincoli di *trust*.

Valutiamo però gli stessi dati in un contesto in cui inseriamo una piccola percentuale di nodi che sono online quasi in ogni momento: negli ultimi anni si è infatti assistito a una crescita esponenziale degli utenti connessi attraverso dispositivi mobili, che rimangono sostanzialmente sempre online nel sistema. La presenza di tali nodi fa diminuire ulteriormente la percentuale minima di nodi che devono essere online per garantire la persistenza dell'informazione nella rete. Con il 5% dei nodi sempre online anche con solo il 27% dei nodi connessi si riesce a garantire la persistenza dei dati di tutti gli utenti in nodi legati ad essi da un legame diretto di amicizia e tale percentuale scende ulteriormente a circa il 23% con il 10% dei nodi sempre online.

Osserviamo infine che i nodi delle nostre reti hanno grado molto inferiore a quello delle social network reali: i dati raccolti ci fanno credere che in molte reti sociali reali, dove il grado medio dei nodi è anche 4 volte superiore a quello dei nodi del nostro modello, sia possibile garantire praticamente sempre la persistenza dell'informazione nella rete senza dover ricorrere al passaggio dei dati a nodi non direttamente collegati al proprietario da un vincolo di amicizia.

Scelta PML iniziali e loro capacità di copertura

Analizziamo ora le caratteristiche dei nodi scelti come PML in una configurazione della rete (poco realistica, ma significativa a livello sperimentale) in cui tutti i nodi si trovano online. Nei nostri diversi esperimenti eseguiti su reti di 1000 nodi estratte con Algorithm 11 vengono eletti come PML dal 6% all' 8% dei nodi, a seconda delle caratteristiche di disponibilità assegnate loro casualmente con il modello di Churn. Questo indica come i

punti di memorizzazione selezionati siano nodi molto centrali nella rete e siano in numero molto limitato.

Valutiamo inoltre la loro capacità di copertura. Partiamo da una configurazione in cui tutti i nodi sono online, in cui ovviamente tutti gli archi risultano coperti e ogni nodo ha eletto il proprio PML. Vogliamo valutare quanti archi rimarrebbero coperti anche dopo la disconnessione di uno qualsiasi dei due nodi che ne costituiscono gli estremi: tali archi sono quelli per cui entrambi i nodi sono stati selezionati come PML oppure esiste un amico comune che è stato selezionato come PML. Dopo l'elezione dei PML, in una configurazione in cui tutti i nodi si trovano online, risulta che 17 361 dei 23 673 archi della rete (pari al 73%) sarebbero comunque coperti dopo la disconnessione di uno dei nodi a essi adiacenti senza la necessità di eseguire l'algoritmo di copertura. Questo mostra che i PML garantiscono una copertura di una porzione molto elevata della rete e permettono quindi a moltissimi nodi di accedere direttamente tramite essi ai dati dei loro vicini. Questo suggerisce inoltre che le molte social cache che, come vedremo nel prossimo paragrafo, devono essere elette in alcuni scenari per garantire la completa copertura della rete in ogni momento, spesso vengono in realtà elette per garantire la copertura di pochi archi. Questo significa che queste social cache devono normalmente memorizzare pochissimi dati o, addirittura, non è necessario che memorizzino alcuna informazione.

Variazione numero PML nel tempo

Il numero di PML eletti nello *Scenario Standard*, come si vede dalla Figura 5.15 che si riferisce ai primi 500 cicli di simulazione, aumenta inizialmente fino a raggiungere un valore massimo di circa 120, per poi stabilizzarsi. Questo è dovuto al fatto che consideriamo una configurazione iniziale in cui tutti i nodi sono online e scelgono il loro PML nello stesso momento. Successivamente la scelta dei PML risente della dinamicità della rete e avviene in momenti diversi per i vari nodi: un nodo n che è un ottimo PML in un certo istante di tempo t, potrebbe non essere più scelto come PML da un nodo che esegue il proprio algoritmo di elezione 30 minuti dopo, perché l'$EstimatedUptime$ di n sarà calato di 30 unità. Nella figura si nota inoltre come la presenza di una percentuale di nodi sempre online renda il numero di PML selezionati ancora più basso e quasi costante in tutti i cicli: gli utenti che sceglieranno un nodo sempre online come proprio PML non dovranno infatti rieleggere mai il proprio PML. La differenza iniziale tra il numero di PML selezionati nei due casi al primo ciclo, in cui tutti i nodi sono online, dipende solo dall'attribuzione casuale della durata media delle sessioni ai nodi e non ha alcun

significato particolare. Infine è importante notare come il numero totale dei PML sia decisamente basso rispetto al totale dei nodi del sistema, il che dimostra che riusciamo a selezionare nodi caratterizzati da buona centralità.

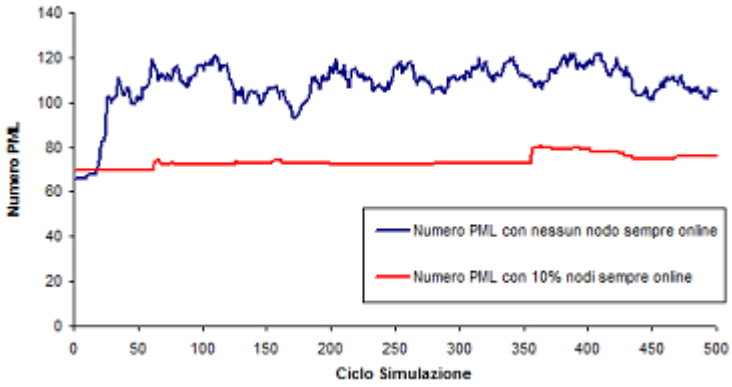


Figura 5.15: Variazione numero PML nel tempo

5.2.4 Copertura in diverse topologie di reti

Andiamo ora ad analizzare il numero di nodi che devono essere selezionati come social cache, in base al numero di utenti connessi, per ottenere una copertura completa di reti di diverse topologie. Come vedremo, il numero di social cache elette sarà fortemente dipendente dalla topologia della rete. Sottolineiamo inoltre che le social cache nella nostra visione sono semplicemente un overlay che garantisce la persistenza dei dati nella rete e che quindi non memorizzano necessariamente dati. Prima di proseguire evidenziamo il primo risultato importante ottenuto con i nostri esperimenti: la rete risulta completamente coperta in ogni istante.

Copertura nello *Scenario Standard*

Il numero di social cache elette nello *Scenario Standard* è piuttosto alto, pari in alcuni momenti alla quasi totalità dei nodi online (si veda Figura 5.16). Questo risultato può apparire a prima vista poco incoraggiante, ma ci sono diverse considerazioni da tenere presenti. Innanzitutto la rete estratta in maniera casuale risulta praticamente priva di cluster, che invece sono presenti nelle reti reali. La presenza di cluster nella rete, come mostreremo più in dettaglio in seguito, fa decrescere in maniera sostanziale il numero di social cache necessarie a ottenere la copertura. In secondo luogo la presenza

nella formula per la scelta dei PML di un termine che tiene conto delle caratteristiche temporali dei nodi non permette di selezionare sempre i nodi che garantiscono una miglior copertura della rete, col vantaggio però di avere dei punti di memorizzazione quanto più possibile stabili nel tempo. Infine un alto numero di social cache non significa affatto che necessitiamo di un alto numero di repliche dell'informazione nella rete, dato che, come mostrato in Figura 5.15, il numero di PML si mantiene sempre molto basso. Come discusso ampiamente in precedenza, le social cache che non sono punto di memorizzazione locale per alcun nodo non devono in molti casi memorizzare alcun dato.

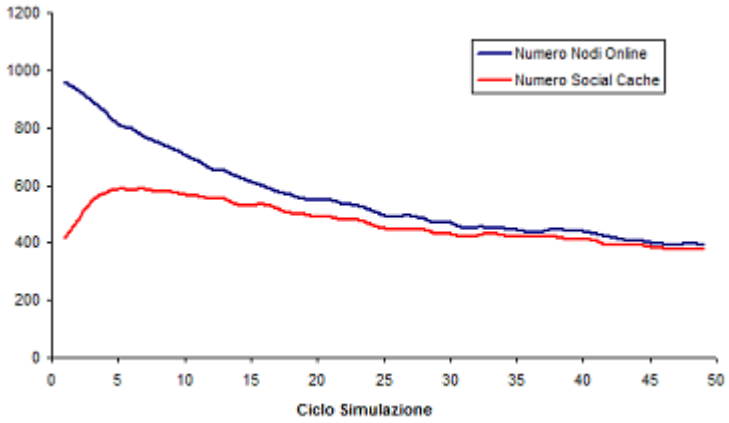


Figura 5.16: Numero nodi online, social cache e PML nello *Scenario Standard*

In definitiva l'obiettivo di mantenere un numero ridotto di social cache totali sembra subordinato alla topologia di rete con cui abbiamo a che fare e per certi grafi sembra possibile garantire la copertura solo eleggendo un alto numero di nodi. Ciò è supportato da un'analisi eseguita in [26] sul numero di social cache necessarie per garantire la copertura di reti di topologie molto diverse. Tale analisi utilizza un concetto di copertura lievemente diverso dal nostro, ma che si basa sulla medesima idea di copertura tramite amici comuni usato nel nostro modello. Il numero di social cache necessario a garantire la copertura di un grafo sociale tratto da Facebook risulta da tale analisi piuttosto alto (pari al 54% dei nodi), ma in altre reti, come un grafo derivato dalle citazioni di autori di pubblicazioni scientifiche o un grafo riferito allo scambio di e-mail in un'azienda, si ottengono risultati molto migliori (pari al 21% e al 13% dei nodi rispettivamente). Questo ci fa quindi ritenere che, in ogni caso, il nostro approccio in topologie di reti diverse possa eleggere un numero di social cache decisamente più basso.

In definitiva il numero di social cache totali da eleggere per garantire la copertura di

un grafo sembra essere strettamente dipendente dalla topologia del grafo stesso.

Elezione in reti molto clusterizzate

La presenza di numerose comunità all'interno delle social network ha reso negli ultimi anni queste reti molto clusterizzate. Purtroppo il dataset su cui lavoriamo per estrarre la rete per i nostri esperimenti risale al 2008, quando Facebook era molto meno clusterizzato di quanto lo sia adesso. Per questo motivo abbiamo condotto alcuni esperimenti per valutare il numero di social cache elette dal nostro algoritmo per garantire la copertura di reti con grado di clusterizzazione maggiore. Per la difficoltà di reperire dataset aggiornati, abbiamo deciso di generare sinteticamente delle reti con diversi gradi di clusterizzazione per analizzare l'impatto che questa caratteristica ha sul numero di social cache necessarie per garantire la completa copertura. Data la natura sintetica delle reti da noi generate, tali misure non costituiscono un punto di paragone affidabile in assoluto ma ci permettono più semplicemente di capire quanto la clusterizzazione di una rete influisca sul numero di nodi necessari per la sua copertura.

Per gli esperimenti è stata creata una rete formata da quattro cluster con alto grado di connettività interna: sono state generate delle clique e sono stati poi "riavvolti" casualmente per ogni nodo fino al 10% dei propri archi in uscita, in modo da non avere una rete troppo regolare.

Sono state quindi fatte tre diverse prove: nelle reti della *Topologia 1*, mostrate su scala ridotta in Figura 5.17a, i cluster della rete sono collegati tra loro da un solo arco. Questa è ovviamente una situazione poco realistica, ma che ci consente di valutare l'importanza di avere una struttura molto clusterizzata.

Nelle reti della *Topologia 2*, più realistiche (vedi Figura 5.17b), il 20% dei nodi stabilisce casualmente fino a 3 collegamenti verso nodi in ognuno degli altri cluster (per un massimo quindi di 9 collegamenti a nodi esterni al cluster per ogni nodo).

Infine nelle reti della *Topologia 3* abbiamo inserito nella struttura delle reti della *Topologia 2* una piccola percentuale di nodi che non fanno parte di alcun cluster, ognuno con grado medio simile a quello degli altri nodi e archi verso tutti gli altri cluster. La struttura di tali reti è mostrata in Figura 5.17c.

I risultati ottenuti per le diverse topologie di reti sono mostrati in Figura 5.18: in configurazioni in cui c'è una frazione di nodi offline, nelle reti della *Topologia 1* vengono elette social cache dal 3% al 7% dei nodi, nelle reti della *Topologia 2* dall'11% al 28% dei nodi e in quelle della *Topologia 3* dal 18% al 32% dei nodi.

In definitiva si può certamente affermare, al di là dei risultati specifici ottenuti per le diverse topologie, che la presenza di cluster all'interno della rete fa diminuire il numero di social cache elette per garantire la copertura del grafo.

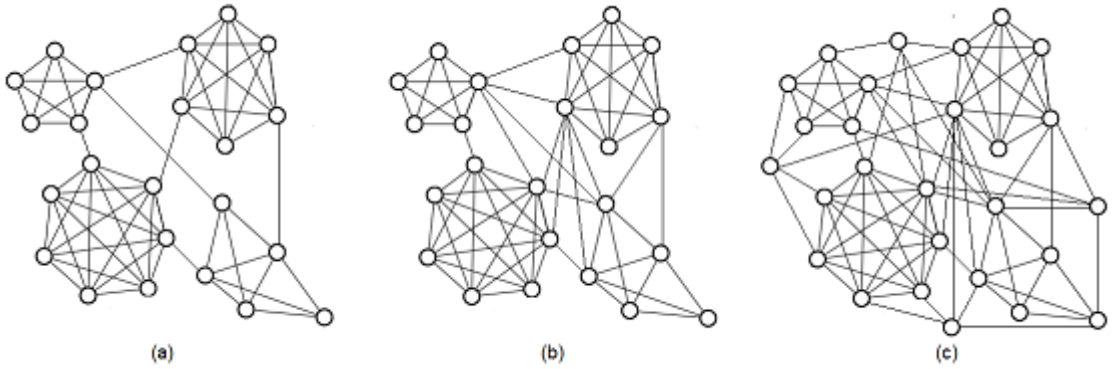


Figura 5.17: Modelli delle topologie di reti clusterizzate generate

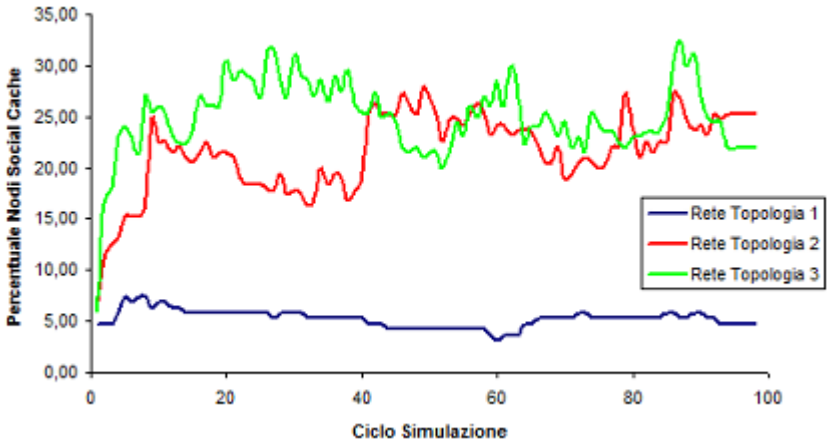


Figura 5.18: Percentuale di nodi eletti social cache nelle diverse topologie di reti con alto grado di clusterizzazione

Capitolo 6

Conclusioni e sviluppi futuri

In questa tesi sono state affrontate le problematiche relative alla gestione delle informazioni sociali di un utente che emergono dalla distribuzione e decentralizzazione del servizio offerto da una OSN attraverso un approccio P2P. È stata definita a supporto del problema una social overlay innovativa, basata sul concetto di ego network proposto da Dunbar in [17]. Una ego network è una rete composta da un utente e da tutti gli utenti che hanno una relazione stabile con esso. Gli studi sociologici condotti da Dunbar hanno mostrato come il numero di utenti con cui un utente ha relazioni stabili sia limitato a 150. Questo modello è inoltre caratterizzato implicitamente da un livello di fiducia: stabilire una relazione con un utente significa infatti avere intrinsecamente fiducia in lui.

Il nostro lavoro ha preso in esame il problema della persistenza dei dati in un overlay P2P basato sul modello sopra descritto. Si è introdotta una strategia, basata sulle caratteristiche di centralità e di disponibilità dei peer del sistema, per individuare nodi della rete che costituiscono dei buoni punti di memorizzazione. In particolare è emersa l'importanza di avere punti di memorizzazione quanto più possibile stabili nel tempo. Da un'analisi su dataset reali è emerso come gli utenti di una OSN siano caratterizzati da durate medie delle sessioni online molto diverse tra loro. Questa osservazione ci ha spinto a scegliere come punti di memorizzazione nodi online caratterizzati da sessioni medie molto lunghe, che hanno maggior probabilità di rimanere connessi a lungo nel sistema. Abbiamo inoltre proposto l'elezione di un overlay dinamico che garantisca in ogni momento la persistenza dei dati della rete all'interno del sistema e assicuri allo stesso tempo il soddisfacimento di alcuni vincoli di *trust*. Proprio per la sua dinamicità l'overlay richiede la definizione di un meccanismo di rielezione ogni qual volta un nodo che ne fa parte passa offline. Tale meccanismo garantisce in ogni momento il più alto grado possibile di persistenza dell'informazione all'interno della rete. Tutti gli algorit-

mi proposti, implementati attraverso un simulatore di reti P2P, hanno dimostrato di raggiungere i loro obiettivi con buoni livelli di performance.

Numerosi sono i possibili sviluppi futuri del nostro lavoro. Innanzitutto le caratteristiche della social overlay Dunbar-based potrebbero essere maggiormente sfruttate nel modello. Nella nostra proposta le ego network sono state utilizzate semplicemente per individuare un certo numero di nodi considerati "fidati". Si potrebbe invece sfruttare il valore del *tie-strength* e la struttura ad anelli concentrici per scegliere come punti di memorizzazione dei dati di un peer nodi dei livelli più interni della sua ego network, che sono caratterizzati da un legame di fiducia più forte con l'utente stesso.

Un altro possibile sviluppo potrebbe consistere nel ridefinire gli algoritmi proposti in un modello in cui le ego network siano rappresentate con grafi orientati. Se pensiamo a Facebook sembra ragionevole associare al concetto di amicizia una relazione bidirezionale: la creazione di un legame di amicizia tra due peer sancisce la stipulazione tra essi di una sorta di contratto in cui affermano di fidarsi l'uno dell'altro. Le cose cambiano però se pensiamo al caso di Twitter: la relazione "essere follower di" è infatti intrinsecamente unidirezionale e sembra quindi ragionevole poter rappresentare le ego network dei peer anche con grafi diretti. Questo richiede però la ridefinizione del nostro concetto di copertura del grafo e di *trust* tra utenti in modo da tener conto del verso degli archi. Questi cambiamenti necessitano di alcune variazioni anche negli algoritmi per l'elezione e la rielezione dei punti di memorizzazione. Sarebbe interessante indagare le differenze tra i modelli proposti e i diversi risultati che si possono ottenere con essi.

Un ulteriore sviluppo futuro riguarda la valutazione del modello su reti di dimensioni maggiori. Purtroppo PeerSim ci permette di valutare il nostro approccio solo su reti di dimensioni piuttosto piccole, che devono essere necessariamente estratte dal dataset complessivo. Implementazioni più articolate potrebbero permetterci di valutare la nostra proposta su reti di dimensioni maggiori e quindi più simili a quelle reali. Sarebbe inoltre interessante valutare le caratteristiche di disponibilità degli utenti delle OSNs su dataset più recenti, che purtroppo non sono attualmente disponibili. In particolare riteniamo che l'evoluzione delle social network porti alla diffusione di reti caratterizzate da un maggior livello di clusterizzazione e da nuove caratteristiche di connettività degli utenti: ad esempio crediamo che la presenza di una sempre più alta percentuale di utenti connessi alla rete attraverso dispositivi mobili come Smartphone o Tablet abbia senz'altro prodotto cambiamenti sostanziali sia nella struttura del grafo sociale che nella dinamicità dei nodi. Riteniamo inoltre che questi cambiamenti producano reti con caratteristiche migliori per il nostro approccio.

Infine lo sviluppo più naturale di questo lavoro è quello riguardante il problema della

effettiva memorizzazione e gestione dei dati. La nostra proposta non si occupa infatti di studiare una tecnica di distribuzione dei dati degli utenti tra le varie social cache, ma garantisce più semplicemente in ogni momento che la persistenza dell'informazione sia possibile e che tutti gli utenti che ne hanno diritto possano accedervi senza violare i vincoli di *trust*. Individua inoltre dei buoni punti di memorizzazione per i dati di ogni singolo utente.

In conclusione crediamo di aver fornito una proposta per la persistenza dei dati in una DOSN basata su un approccio innovativo e, senza avere la pretesa di aver affrontato in maniera esaustiva tutti i problemi collegati all'argomento, speriamo che essa possa rappresentare un originale punto di partenza per una proposta più completa che analizzi più a fondo alcune delle questioni che rimangono ancora aperte.

Bibliografia

[1] S. Acharya and S. B. Zdonik. An Efficient Scheme for Dynamic Data Replication. Technical report, Providence, RI, USA, 1993.

[2] V. Arnaboldi, M. Conti, A. Passarella, and F. Pezzoni. Analysis of Ego Network Structure in Online Social Networks. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Conference on Social Computing (SocialCom)*, pages 31–40, 2012.

[3] V. Arnaboldi, A. Guazzini, and A. Passarella. Egocentric Online Social Networks: Analysis of Key Features and Prediction of Tie Strength in Facebook. *Computer Communications*, 36(10-11):1130–1144, 2013.

[4] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in p2p systems. *Commun. ACM*, 46(2):43–48, 2003.

[5] A.-L. Barabasi and R. Albert. *Science*, (5439):509–512.

[6] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *Proceedings of IPTPS'03*, 2003.

[7] S. Blond, F. Fessant, and E. Merrer. Finding Good Partners in Availability-Aware P2P Networks. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS '09, pages 472–484, Berlin, Heidelberg, 2009.

[8] A. Boutet, A.-M. Kermarrec, E. Le Merrer, and A. Van Kempen. On the Impact of Users Availability in OSNs. In *Proceedings of the Fifth Workshop on Social Network Systems*, SNS '12, pages 4:1–4:6, New York, NY, USA, 2012.

[9] D. M. Boyd and N. B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230, 2007.

[10] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta. PeerSoN: P2P social networking – early experiences and insights. In *Proc. ACM Workshop on Social Network Systems*, 2009.

[11] S. Y. Chan, I. X. Y. Leung, and P. Liò. Fast centrality approximation in modular networks. In *CNIKM '09: Proceeding of the 1st ACM international workshop on Complex networks meet information and knowledge management*, pages 31–38, 2009.

[12] M. Conti, A. Passarella, and F. Pezzoni. A Model to Represent Human Social Relationships in Social Network Graphs. In *Social Informatics*, volume 7710 of *Lecture Notes in Computer Science*, pages 174–187. 2012.

[13] L. A. Cutillo, R. Molva, and T. Strufe. Safebook: a Privacy Preserving Online Social Network Leveraging on Real-Life Trust. *Communication Magazine, IEEE*, 47(12), 2009.

[14] A. Datta, S. Buchegger, L.-H. Vu, T. Strufe, and K. Rzadca. Decentralized Online Social Networks. In *Handbook of Social Network Technologies and Applications*, pages 349–378. 2010.

[15] A. De Salve. Persistenza dei dati in P2P Dunbar-based Social Overlay. Master's thesis, Università di Pisa, 2013.

[16] J. R. Douceur. Is Remote Host Availability Governed by a Universal Law? *SIGMETRICS Perform. Eval. Rev.*, 31(3):25–29, 2003.

[17] R. I. M. Dunbar. The social brain hypothesis. *Evolutionary Anthropology: Issues, News, and Reviews*, 6(5):178–190, 1998.

[18] P. Erdős and A. Rényi. On the Evolution of Random Graphs. In *Publication of the Mathematical Institute of Hungarian Academy of Sciences*, pages 17–61, 1960.

[19] E. Gilbert and K. Karahalios. Predicting tie strength with social media. In *In Proceedings of the Conferece on Human Factors in Computing Systems (CHI'09)*, page 220, 2009.

[20] D. Gkorou, J. Pouwelse, and D. Epema. Betweenness Centrality Approximations for an Internet Deployed P2P Reputation System. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1627–1634, 2011.

[21] S. Golder, D. Wilkinson, and B. Huberman. Rhythms of Social Interaction: Messaging Within a Massive Online Network. In *Communities and Technologies 2007*, pages 41–66. 2007.

[22] R. Gracia-Tinedo, M. Sanchez Artigas, and P. Garda Lopez. Analysis of data availability in F2F storage systems: When correlations matter. In *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*, pages 225–236, 2012.

[23] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz. LifeSocial.KOM: a secure and P2P-based solution for online social networks. In *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, pages 554–558, 2011.

[24] L. Gyarmati and T. Trinh. Measuring User Behavior in Online Social Networks. *Network, IEEE*, 24(5):26–31, 2010.

[25] L. Han, B. Nath, L. Iftode, and S. Muthukrishnan. Social Butterfly: Social Caches for Distributed Social Networks. In *Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE third international conference on*, pages 81–86, 2011.

[26] L. Han, M. Punceva, B. Nath, S. Muthukrishnan, and L. Iftode. SocialCDN: Caching techniques for distributed social networks. In *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*, pages 191–202, 2012.

[27] P. Holme and J. Saramäki. Temporal networks. *Physics Reports*, 519(3):97–125, 2012.

[28] D. P. John and J. Harrison. A Distributed Internet Cache. In *Proceedings of the 20th Australian Computer Science Conference*, pages 5–7, 1997.

[29] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(11):P11005, 2011.

[30] M. La Gala, V. Arnaboldi, M. Conti, and A. Passarella. Ego-net digger: a new way to study ego networks in online social networks. In *Proceedings of the First ACM International Workshop on Hot Topics on Interdisciplinary Social Networks Research*, HotSocial '12, pages 9–16, New York, NY, USA, 2012.

[31] K. A. Lehmann and M. Kaufmann. Decentralized algorithms for evaluating centrality in complex networks. Technical report, Universitätsbibliothek Tübingen, 2003.

[32] D. Liu, A. Shakimov, R. Cáceres, A. Varshavsky, and L. P. Cox. Confidant: protecting OSN data without locking it up. In *Proceedings of the 12th ACM/IFIP/USENIX international conference on Middleware*, Middleware'11, pages 61–80, Berlin, Heidelberg, 2011.

[33] P. V. Marsden. Egocentric and sociocentric measures of network centrality. *Social Networks*, 24(4):407 – 422, 2002.

[34] P. V. Marsden and K. E. Campbell. Measuring Tie Strength. *Social Forces*, 63(2):482–501, 1984.

[35] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. pages 53–65, 2002.

[36] G. Mega, A. Montresor, and G. Picco. Efficient Dissemination in Decentralized Social Networks. In *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, pages 338–347, 2011.

[37] J. W. Mickens and B. D. Noble. Exploiting Availability Prediction in Distributed Systems. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 6–6, Berkeley, CA, USA, 2006.

[38] A. Montresor and M. Jelasity. PeerSim: A Scalable P2P Simulator. In *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, pages 99–100, 2009.

[39] R. Narendula, T. Papaioannou, and K. Aberer. A Decentralized Online Social Network with Efficient User-Driven Replication. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Conference on Social Computing (SocialCom)*, pages 166–175, 2012.

[40] M. E. J. Newman. Assortative Mixing in Networks. *Phys. Rev. Lett.*, 89:208701, 2002.

[41] M. E. J. Newman. The structure and function of complex networks. *SIAM REVIEW*, 45:167–256, 2003.

[42] S. G. B. Roberts, R. I. M. Dunbar, T. V. Pollet, and T. Kuppens. Exploring variation in active network size: Constraints and ego characteristics. *Social Networks*, 31(2):138–146, 2009.

[43] D. Schiöberg, F. Schneider, G. Trédan, S. Uhlig, and A. Feldmann. Revisiting Content Availability in Distributed Online Social Networks. *CoRR*, abs/1210.1394, 2012.

[44] R. Sharma and A. Datta. SuperNova: Super-peers Based Architecture for Decentralized Online Social Networks. *CoRR*, abs/1105.0074, 2011.

[45] A. Sutcliffe, R. Dunbar, J. Binder, and H. Arrow. Relationships and the social brain: Integrating psychological and evolutionary perspectives. *British Journal of Psychology*, 103(2):149–168, 2012.

[46] J. Tang, M. Musolesi, C. Mascolo, and V. Latora. Temporal distance metrics for social network analysis. In *Proceedings of the 2nd ACM workshop on Online social networks*, WOSN '09, pages 31–36, New York, NY, USA, 2009.

[47] K. Wehmuth and A. Ziviani. Distributed Assessment of Network Centralities in Complex Social Networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*, pages 1046–1049, 2012.

[48] C. Wilson, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. Beyond Social Graphs: User Interactions in Online Social Networks and Their Implications. *ACM Trans. Web*, 6(4):17:1–17:31, 2012.

[49] Z. Yao, D. Leonard, X. Wang, and D. Loguinov. Modeling Heterogeneous User Churn and Local Resilience of Unstructured P2P Networks. In *In ICNP*, pages 32–41, 2006.