Privacy in DOSN:
Un approccio basato su controllo degli accessi

Alessio Caporale
mat: 411017

Filippo Delfino
mat: 428921

Relatori
Prof.ssa Laura Ricci
Dott. Paolo Mori

Corso di Laurea Magistrale in Informatica
Università degli Studi di Pisa

Anno Accademico

2014/2015

Indice

Elenco delle figure

Capitolo 1

Introduzione

La diffusione su scala mondiale di Internet ha prodotto cambiamenti significativi nella vita di tutte le persone, portando benefici in diversi campi come quello tecnologico, scientifico e sociale. Il fenomeno più significativo negli ultimi dieci anni risulta, senza dubbio, essere la diffusione delle reti sociali: **Online Social Network** (*OSNs*) [42]. Le *OSNs* sono servizi che permettono agli utenti la costruzione di un profilo e la possibilità di interconnettersi con gli altri iscritti. Nate con lo scopo di permettere agli utenti di mantenersi in contatto, ignorando le distanze geografiche, sono diventate popolari con l'avvento di Facebook, a tal punto da rappresentare un'ampia porzione del tempo trascorso dagli utenti online.

Una delle principali problematiche nelle *OSNs* è rappresentata dalla gestione della privacy degli utenti. L'enorme quantità di dati sensibili, tra cui nomi, numeri di telefono e interessi, pubblicata dagli utenti deve essere "difesa" dagli attacchi esterni.

La maggior parte delle *OSNs*, invece, richiedono agli utenti, prima che utilizzino il sistema, di acconsentire a un insieme di termini contrattuali, in cui vengono presentate delle clausole grazie alle quali le *Social Network* possono salvare i contenuti dell'utente (anche una volta eliminato il profilo) o addirittura condividerli con terze parti. Ad esempio, i contenuti personali dell'utente possono essere utilizzati da software che incorporano informazioni per personalizzare le applicazioni o da agenzie pubblicitarie per indirizzare le pubblicità ad un pubblico che più facilmente sarà interessato ai prodotti.

Il problema deriva dalla presenza delle entità centralizzate che archiviano i dati e ne hanno il totale controllo. Allo scopo di eliminare la dipendenza da queste entità, sono state sviluppate le **Distributed Online Social**

Networks (*DOSNs*) [40], servizi sociali completamente decentralizzati e distribuiti. L'architettura delle *DOSN* è basata sul paradigma *Peer-to-Peer* [41]: sono gli utenti stessi ad avere sia il ruolo di consumatore che di gestore del servizio. L'utente ha il controllo sui propri contenuti per tutta la durata delle sue sessioni online.

Questa tipologia di reti sociali risulta avere una topologia di rete molto dinamica. La continua entrata e uscita di utenti (*churn*) dalla rete può portare a diverse problematiche:

Data Availability: le informazioni di ogni utente devono essere disponibili nel sistema anche durante i periodi di inattività degli stessi.

Availability Prediction: studio della disponibilità degli utenti.

Privacy: fornire all'utente metodologie efficienti di controllo degli accessi ai contenuti; infatti, anche se non è presente un entità centralizzata, occorre che ogni utente fornisca i propri contenuti solo agli utenti autorizzati, secondo le politiche di privacy definite.

In letteratura sono stati presentati diversi esempi di *DOSNs*, tra le più importanti troviamo:

- **Diaspora**[1]: i contenuti sono gestiti da utenti speciali (*pod*) che svolgono la funzione di server decentralizzati.
- **PeerSon** [14]: offre un servizio di scambio diretto dei dati, l'architettura sfrutta una *DHT* per il servizio di *look-up* di utenti e contenuti.
- **Safebook** [15]: composto da tre componenti: un servizio per l'acquisizione dei certificati necessari per unirsi alla rete, le *Matrioska* per la gestione degli amici e una *DHT* per il *look-up*.
- **SuperNova** [16]: basta sul concetto di *Super-Peer*: utenti con maggiori funzionalità di servizio.
- **LifeSocial.KOM** [17]: le funzionalità sono implementate come *plug-in*, utilizza una *DHT*.
- **Confidant** [18]: basato su un architettura *P2P*, alcune informazioni salvate in server *cloud*.

[1] https://joindiaspora.com

Una caratteristica che accomuna tutti gli approcci elencati è l'utilizzo pesante della **crittografia**. L'operazione di cifratura dei contenuti è la modalità più semplice per garantire la protezione durante lo scambio delle informazioni, consente la memorizzazione delle informazioni anche in nodi untrusted e la prevenzione di attacchi informatici; tuttavia, tale procedura risulta essere dispendiosa da un punto di vista computazionale.

Questa tesi si prefigge di definire un'architettura distribuita per reti sociali che fornisca un **controllo degli accessi** ai contenuti che eviti, nella maggior parte dei casi, l'uso della cifratura. L'architettura è basata sulla definizione di una *friend-to-friend overlay network* in cui ogni nodo apre connessioni dirette con i suoi amici. Le connessioni presenti sull'overlay corrispondono quindi a relazioni di trust reali tra amici. Il problema su cui si focalizza la tesi è quello della definizione di un sistema di controllo degli accessi che consenta:

- quando l'utente è online, di verificare se la richiesta di accesso effettuata da un amico può essere soddisfatta.
- quando l'utente va offline, di scegliere in base alle politiche di sicurezza, l'amico o gli amici su cui salvare i propri contenuti.

Infatti, a seguito della disconnessione di un utente dal sistema, i contenuti vengono **replicati** e assegnati, fra le sue amicizie, ai nodi più adatti. Un contatto C è ritenuto adatto a ricevere il profilo se, in primo luogo, C soddisfa le politiche di privacy del nodo che si disconnette e se ha delle buone prestazioni legate al comportamento delle sue sessioni nel *social network* e alla banda a disposizione.

I riferimenti ai nodi che contengono i contenuti di un utente non connesso al servizio sono salvati in una **Distributed Hash Table** (*DHT*) [45]. Un nodo tramite la *DHT* effettua la ricerca di un contatto; se esso non è online la ricerca restituisce il riferimento agli utenti che contengono le repliche delle informazioni cercate. I dati degli utenti, quindi, sono sempre presenti nel sistema evitando, nella maggior parte dei casi, l'uso del processo di cifratura dei contenuti, che risulta essere ancora più gravoso computazionalmente nei casi di richieste di accesso particolarmente specifiche.
Solo nel caso in cui l'utente, al momento della disconnessione, non abbia amici online che verificano le politiche da lui definite, per garantire che i contenuti non vengano acceduti da contatti non autorizzati, i contenuti vengono salvati su un nodo untrusted di una *DHT*.

In questa tesi abbiamo sviluppato il modulo che si occupa della generazione delle politiche di privacy e della gestione delle richieste di accesso ai contenuti da parte degli utenti. Le politiche di privacy rappresentano le condizioni che devono essere verificate dall'utente che richiede l'accesso ad un determinato contenuto.

L'architettura del modulo implementata si basa sull'uso di **eXtensible Access Control Markup Language** (*XACML*) [34], il linguaggio, basato su *XML*, con cui sono state definite le politiche e le richieste. L'espressività del linguaggio *XACML* permette di specificare un gran numero di politiche, come ad esempio: accessi basati sul grado delle relazioni di amicizia, sulla condivizione di interessi, sulla distanza geografica e altro.

In primo luogo, sono state implementate un insieme di *policy* considerando il profilo degli utenti come un'entità unica, successivamente si è estesa la definizione delle politiche considerando un **profilo strutturato in maniera gerarchica**. Il profilo è suddiviso in macro-categorie, a loro volta divise in categorie e singoli contenuti. Grazie a questo approccio, è possibile, al momento dell'uscita di un utente dal servizio, partizionare i suoi contenuti in modo tale da ridurre (o eliminare completamente) la necessità di cifrare questi ultimi; ciò è reso possibile dal fatto che, nel caso in cui nessun nodo abbia accesso all'intero profilo del nodo disconnesso, più amici, che hanno legittimo accesso a diverse parti del profilo, si prendano carico di tali informazioni.

Nel Capitolo 2 saranno descritte le principali *OSN* e *DOSN* e come queste affrontano il problema della privacy. Saranno, inoltre, presentate le principali caratteristiche delle reti complesse e in che modo è possibile definire un'analisi formale di queste reti. Successivamente, nel Capitolo 3, sarà esposta l'architettura **DOP** (*Distributed Online social network Privacy*) proposta in questa tesi, presentando i moduli sviluppati. Il Capitolo 4 descriverà nel dettaglio l'implementazione di tali moduli. Nel Capitolo 5 sarà esposta una analisi dei risultati ottenuti attraverso i test, infine il Capitolo 6 presenterà le conclusioni e gli sviluppi futuri.

Capitolo 2

Stato dell'Arte

In questo capitolo verrà trattata la letteratura relativa ai contenuti sviluppati nella tesi. In particolare saranno illustrate le attuali proposte di tecniche per lo sviluppo di *social network*, sia centralizzate che decentralizzate, presentati sia in ambito accademico che commerciale.

Si inizierà illustrando cos'è un'*Online Social Network*, in particolare l'aspetto legato alla gestione della privacy; a tale riguardo, saranno presentati alcuni meccanismi per il controllo degli accessi. A questo punto, si passerà a descrivere le *Distributed Online Social Network*, evidenziando le problematiche e gli esempi più noti di questa tipologia di rete sociale.

Successivamente, verranno trattate alcune tecniche relative all'analisi delle reti complesse ed utilizzate per lo studio delle reti sociali; verrà descritto, infine, il protocollo *Pastry*, la *DHT* utilizzata nel nostro *framework* e le tecniche crittografiche utilizzate al fine di preservare l'integrità e la confidenzialità dei contenuti nelle reti.

2.1 Online Social Network (OSN)

Negli ultimi anni, nell'ambito delle relazioni sociali vengono largamente utilizzate le cosiddette *Online Social Networks* (*OSNs*).
Una *OSN* è una rete strutturata di rapporti sociali, una trama di relazioni che tiene unite più persone all'interno dello stesso sistema.
Esistono diversi esempi di *OSNs*, ognuno con le proprie peculiarità, ma con tratti fondamentali condivisi. In una *OSN* l'**utente** è rappresentato come un'entità digitale tramite il suo profilo, definisce le relazioni con altri utenti,

con i quali può interagire e condividere contenuti.

Mentre alcune *OSNs* sono costruite per agevolare le relazioni lavorative, la maggior parte ha come scopo lo sviluppo delle interazioni personali, svago, organizzazione di eventi e diffusione di notizie.
La popolarità crescente di questo tipo di servizio tra gli utenti di tutto il mondo, ha portato allo sviluppo, negli ultimi anni di un gran numero di nuove piattaforme; tra le più utilizzate ricordiamo: *Facebook*, *Twitter*, *Linkedin* e *Google+*.

Le *OSN* citate si basano su un'architettura **centralizzata**: le interazioni fra gli utenti, lo storage dei contenuti dei profili e tutte le funzionalità sono gestiti da un insieme di server che rappresentano punti di centralizzazione.
Una gestione del sistema di questo genere ha diversi benefici, tra cui la semplicità della gestione della persistenza dei dati e della loro informazioni. Se però il livello di sicurezza non è adeguato, il sistema può mettere a rischio i contenuti degli utenti. Inoltre, la gestione dei profili affidata a singoli punti di controllo fa sorgere problemi di *fault tolerance* (disponibilità dei dati in caso di crash dei server) e di *performance* (colli di bottiglia).

Un esempio significativo di *online social network* centralizzata molto diffuso al momento è **Facebook**, che useremo come punto di riferimento per molti aspetti. Le relazioni tra gli utenti su Facebook sono rappresentate dalle **amicizie**, mentre le relazioni tra utenti ed altre entità possono essere rappresentate da: "**mi piace**", **commenti** e **tag**. Ogni utente può definire fino a 5000 amicizie e può offrire il proprio mi piace a pagine, status, fotografie ed altro ancora.

Qualsiasi azione esercitata sull'*online social network* viene registrata e divulgata agli utenti autorizzati a vedere l'azione stessa. Quindi, se un utente pubblica un aggiornamento di status, è consapevole che il contenuto stesso risulta accessibile da chiunque (nel caso in cui si scelga una visibilità "pubblica"), oppure solo dai propri amici, dagli amici dei propri amici, da specifici utenti o da specifiche liste di contatti.

Facebook mantiene uno storico di tutti gli status caricati all'interno del **Diario** ("Timeline"), pagina sulla quale è così possibile scorrere nel tempo quanto archiviato dall'utente sul sistema.

Grazie alle **notifiche**, Facebook segnala ad ogni utente eventuali messaggi dei propri amici più stretti, oppure eventuali commenti o "mi piace" ai propri aggiornamenti antecedenti. Le notifiche sono uno degli aspetti che più legano l'utente a Facebook, poiché consigliano con frequenza la visita del

social network alla ricerca di nuove interazioni con i propri amici.

La **condivisione** è una ulteriore possibilità a disposizione degli utenti, poiché consente di rilanciare quanto pubblicato da un altro utilizzatore dando così maggior visibilità al contenuto. La condivisione è qualcosa di più di un semplice "mi piace", poiché consente di far proprio il contenuto, di divulgarlo ai propri contatti ed al contempo di render credito alla fonte originaria della pubblicazione.

Facebook nasce nel 2004 come semplice bacheca di utenti in ambito universitario. La mente a capo del progetto è quella di Mark Zuckerberg, che, partito da un semplice network per il proprio campus (Harvard), ha dato vita a quella che è oggi la più *grande community* online di sempre. La storia di Facebook presenta alti e bassi, con vari problemi legali che contraddistinguono i tentativi del gruppo di trasformare il crescente successo raccolto in un sempre più redditizio modello di business. Gli scontri sulla privacy e su annunci oltremodo invasivi non hanno però mai rallentato la crescita del *social network*, la cui ascesa ha iniziato a calmierare il proprio ritmo soltanto a fronte di un mercato potenziale sempre più piccolo ed al cospetto di una concorrenza sempre più forte e differenziata.

Nel tempo sono stati molti gli esperimenti approntati che sono andati in generale sempre in due direzioni: il miglioramento dell'esperienza dell'utente, pur cercando di raccogliere quanti più dati personali possibile, e la ricerca di un modello di business sempre più solido e profittevole.

Al termine del suo primo anno di attività, Facebook contava già **1 milione di iscritti**. Nell'ottobre del 2012 è raggiunta e superata quota 1 miliardo di utenti. Nel 2014, Facebook conta ormai 1,23 miliardi di utenti attivi, dove per "utente attivo" si intende un utente iscritto a Facebook che nell'ultimo mese ha in qualche modo interagito con il *social network*. Il numero va inteso per quel che è: persone reali che hanno effettuato il login, oppure hanno cliccato su uno dei pulsanti *ad hoc* ospitati su siti terzi. Il numero rappresenta non tanto il volume degli iscritti, quanto più il peso delle loro interazioni.

Tutti i giorni vengono cliccati circa 6 miliardi di "mi piace" e ci sono 25 milioni di Pagine attive in ambito PMI (Piccole e Medie Imprese): le pagine sono bacheche sulle quali non sono le persone, ma soprattutto associazioni ed aziende, a portare online le proprie informazioni.

La rete è una fitta trama di relazioni che, dall'inizio delle attività di Facebook ad oggi, conta 201,6 miliardi di connessioni: è questa rete a tenere in piedi l'intera struttura, poiché le relazioni sono il nesso su cui viaggiano status e notifiche, coinvolgimento e identità.

2.1.1 Gestione della Privacy nelle OSN

Nel tentativo di proteggere i contenuti personali degli utenti, le *OSN* adottano semplici approcci alla gestione degli accessi incentrati sull'utente, dove l'utente stesso ha la possibilità di specificare quale politica utilizzare nell'accesso delle sue risorse.

La maggior parte delle *OSN* implementa la gestione delle politiche di privacy in maniera estremamente semplice, fornendo all'utente la possibilità di impostare le proprie informazioni personali come *pubbliche*, *private* o *accessibili ai contatti diretti*.

OSN	Purpose	Protection Options
Bebo	general	public, private, 1st-degree contacts, selected contacts
Google+	general	public, private, 1st-degree contacts, selected contacts
Facebook	general	public, private, 1st and 2nd degree contacts, selected contacts
Friendster	general	private, 1st-degree contacts, users from selected continents
MySpace	general	public, private, 1st-degree contacts, members > 18 years old
Multiply	general	public, private, 1st and n degree contacts, selected contacts
Flickr	photos	public, private, 1st degree contacts
Last.fm	music	public, private, 1st degree contacts
Xing	business	public, private, selected contacts
LinkedIn	business	public, private, selected members

Figura 1. Tabella riassuntiva sulle opzioni di protezione dei vari OSN.

In Figura 1 [1] sono presenti il contesto in cui si pongono diverse *social network* e le opzioni per la specifica delle privacy che esse presentano. Oltre alle impostazioni base descritte, ogni sistema mette a disposizione un insieme limitato di scelte, che tipicamente permettono di partizionare il profilo in maniera pubblica o privata.

Alcune di queste *OSNs*[1] permettono di definire vere e proprie regole, mediante le quali è possibile decidere in maniera specifica quali liste o gruppi di utenti possano accedere ai contenuti. Altri sistemi centralizzati[2] non per-

[1]Bebo (http://www.bebo.com/), Multiply (http://multiply.com/) e Xing (http://www.xing.com/), Facebook (http:// www.facebook.com), Google+ (https://plus.google.com/) e LinkedIn (http://www.linkedin.com/).

[2]MySpace (http://myspace.com/) e Last.fm (http://www.last.fm/).

mettono la suddivisione dell'insieme dei contatti in gruppi, ma semplicemente rendono possibile la condivisione a tutti gli amici (1°grado), amici di amici (2°grado) fino ad un grado di amicizia definito dall'utente.

Nei sistemi centralizzati, in generale, è possibile creare un solo tipo di relazione: l'amicizia, tuttavia, recentemente alcune *OSN*, come Facebook e Google+, hanno incluso la possibilità di creare tipi differenti di relazione specificando se il contatto aggiunto appartiene alla famiglia, al lavoro, etc. LinkedIn e Multiply hanno una scelta più varia e specializzata nel campo del lavoro, tra i tipi di relazioni sono, infatti, presenti: compagno di classe, partner d'affari e collega.

Gli autori in [2] propongono un sistema con meccanismo di controllo degli accessi basato sulla fiducia, in cui ogni utente ha un valore di reputazione calcolato in base al valore di fiducia assegnatogli dagli altri utenti del sistema. Ad ogni risorsa viene assegnato un livello di confidenza pari o inferiore al grado di fiducia dell'utente che la possiede. Gli utilizzatori del sistema possono accedere solamente ai contenuti che hanno un livello di confidenza inferiore o uguale alla loro reputazione. Il controllo sull'accesso è rafforzato utilizzando la crittografia a chiave simmetrica.

In [3] viene proposto il sistema **D-FOAF**, in cui ogni relazione è associata ad un livello di fiducia e l'utente definisce le politiche di accesso in base al valore di fiducia e al cammino di massima lunghezza che mette in comunicazione il proprietario e il richiedente. Viene definito una ontologia basata sul concetto *Friend of Friend*. Il sistema può essere esteso considerando il caso in cui non si ha solo un tipo di relazione, come mostrano gli autori in [4].

Di grande interesse sono i lavori in questo campo effettuati da Carminati et al. [1, 5] in cui viene proposto un meccanismo di accesso per le *OSN* basato su regole, dove gli utenti vengono autorizzati tenendo presente il tipo, la robustezza e la fiducia delle relazioni esistenti.

Le informazioni riguardanti le relazioni vengono conservate in un'architettura semi-decentralizzata, codificate in certificati e salvati in server appositi. L'accesso alla risorsa avviene solamente se il richiedente dimostra che la relazione fra lui e chi detiene la risorsa possiede le proprietà necessarie. Il possessore del contenuto può designare dei co-possessori, i quali hanno il potere di modificare la politica di accesso della risorsa tramite un sistema di votazione.

Lockr è un sistema proposto in [6] che utilizza le relazioni presenti nella *OSN* per definire le politiche di privacy. L'accesso ai dati è basato sui certificati delle chiavi pubbliche, il richiedente deve essere in possesso della chiave

pubblica con i permessi necessari ad accedere ai dati. Il sistema ha come focus la definizione di policy sulle singole risorse.

Squicciarini et al. [7] propone un approccio alle *policy* collaborativo per i sistemi che specificano le politiche di privacy in base al massimo grado di profondità nelle amicizie. Gli utenti specificano quali altri nodi della rete possano mantenere i loro dati, tali nodi prenderanno parte alla votazione in cui si deciderà quale politica applicare su ogni informazione. Il sistema è implementato in modo da premiare i possessori originali dei dati che includono altri utenti e gli utenti che scelgono correttamente la politica di privacy. Allo scopo di ridurre la frequenza delle votazioni, è stato incluso un sistema di adattamento che applica le stesse politiche a contenuti simili.

In [8] gli autori hanno sviluppato un sistema automatico di negoziazione delle politiche di privacy, volto a stabilire delle politiche accettabili sia dal client (utente) che dal servizio di collezione dei dati. Ad ogni contenuto vengono associati tre tipi di metadati: il ricevente, lo scopo d'uso del contenuto e la politica di conservazione dei dati del server. Ai tre tipi di metadati l'utente assegna uno di questi tre valori: ideale, accettabile e non accettabile, similmente il server definisce la sua politica di privacy, ma basandosi sul dato che deve collezionare. Durante la fase di negoziazione, l'utente e il server cercano di trovare politiche, per tutti i dati dell'utente, che abbiano almeno valore accettabile per entrambe le parti.

Baker et al. [23], al fine di aiutare i ricercatori nella classificazione e nel confronto di differenti politiche di privacy, hanno introdotto una loro tassonomia.

Secondo questa tassonomia, gli elementi che caratterizzano una politica di privacy sono i seguenti:

Scopo Rappresenta il motivo per il quale un utente mette a disposizione uno specifico dato in modo che altri utenti possano usufruirne legittimamente. Lo scopo può essere definito per un uso singolo o multiplo della risorsa.

Visibilità Definisce chi ha il permesso di accedere in maniera legittima al dato condiviso, è l'elemento cruciale che deve rendere inaccessibile l'informazione a chi non ha i diritti.
Secondo la tassonomia la visibilità dei dati può essere ristretta al solo utente in possesso della risorsa, alla rete, agli amici o agli amici di amici, a terze parti oppure visibile universalmente.

Granularità Definisce il grado di precisione con cui l'informazione deve essere rivelata al momento dell'accesso di un utente. Se il dato viene restituito interamente la granularità è definita come specifica, se, invece, la risorsa risulta essere composta da più parti la granularità può essere parziale. Inoltre, il grado di precisione può essere impostato come esistenziale, in tal caso all'utente che effettua la ricerca viene rivelato il dato solo se quest'ultimo esiste oppure no.
Settare correttamente la granularità è importante soprattutto per la gestione di dati composti da parti sensibili.

Persistenza Indica il periodo di tempo in cui è possibile accedere ad una data informazione, nella tassonomia tale parametro può essere impostato tramite una data oppure, se non si vuole specificare un periodo particolare, tramite ∞.

Purpose	Visibility	Granularity
RSm=Reuse Same	H=House	S=Specific
RS=Reuse selected	F=Friends	P=Partial
RA=Reuse Any	FoF=Friends of Friends	
A=Any	N=Network	
	AW=All/World	

Figura 2. Politiche applicabili per ogni elemento e legenda acronimi.

In Figura 2 è presente una legenda che verrà utilizzata per la classificazione delle *OSNs*. Lo scopo può assumere i valori: riuso per il solito scopo, riuso solo per scopi specifici, riuso per qualsiasi scopo oppure nessuna restrizione. La visibilità assume i valori: locale, amici, amici di amici, rete o senza restrizioni. La granularità può essere riferita solo al contenuto specifico o ad un agglomerato di dati.

Le politiche di *privacy* della maggior parte dei *social network* suddividono i dati in quattro diversi livelli, che differiscono in base allo scopo per cui sono stati messi a disposizione. I dati potrebbero far parte di più livelli e talvolta le politiche potrebbero essere vaghe per ciò che concerne lo scopo di una certa informazione.

Registrazione Le informazioni sensibili necessarie ad identificare l'utente tra tutti gli utilizzatori del sistema.

Networking Dati relativi alle relazioni intraprese dall'utente nei confronti degli altri utilizzatori del sistema.
Ogni volta che viene aggiunto un elemento nella rete dell'utente, vengono generate informazioni relative a questo livello.

Contenuto I dati veri e propri con cui gli utenti interagiscono all'interno del *social network*, solitamente, per questo livello di informazioni, è possibile settare il livello di *privacy* voluto utilizzando le impostazioni di sistema.

Attività Informazioni che derivano dai log dei *web server*, dai *cookie* o da ulteriori mezzi per tenere traccia delle attività dell'utente.

Lo scopo di [24] è quello di analizzare il comportamento di sei famosi *social network*, in base alla tassonomia descritta.

Si analizzano separatamente scopo, granularità e visibilità dei seguenti *social*: Facebook, MySpace, Twitter, LinkedIn, YouTube e Orkut.

2.1.2 Analisi Scopo

In generale si nota come le *OSN* acquisiscono dati per ogni scopo o per uno scopo preciso che verrà riutilizzato in seguito per ogni scopo.

Le informazioni di livello registrazione, essendo dati molto sensibili, sono protette dalle politiche delle reti analizzate e non verranno rilasciati facilmente come il resto delle risorse.

I dati relativi ai livelli di contenuto e attività non sono vincolati a seconda dello scopo, quindi per tutti e sei i *social network* analizzati tali informazioni possono essere usate per qualsiasi fine. È possibile che anche nel livello di attività ci siano dati sensibili per l'utente, per questo motivo Twitter e LinkedIn permettono la diffusione delle informazioni solo per scopi molto specifici.

Purpose	LinkedIn	Twitter	Orkut	Facebook	MySpace	YouTube
Registration	RA	RA	RSm	RA	RS	RA
Networking	A	A	A	RA→A	A	A
Content	A	A	A	RA→A	A	A
Activity	RS	RS	A	RA	A	A

Figura 3. Politiche di scopo per i vari OSN.

2.1.3 Analisi Visibilità

Le informazioni di registrazione chiaramente non potranno essere visibili a nessuno se non all'utente stesso. Allo stesso modo anche i dati relativi alle attività dell'utente non devono essere visibili da altre entità.

La visibilità delle informazioni dei livelli di contenuto e networking in genere viene decisa dall'utente stesso nel momento in cui specifica le politiche da utilizzare; in YouTube, invece, tutte le informazioni sono pubbliche.

Visibility	LinkedIn	Twitter	Orkut	Facebook	MySpace	YouTube
Registration	H	H	H	H	H	H
Networking	F→AW	F→AW	F→N	H→N	F→AW	AW
Content	F→AW	F→AW	F→N	H→N	F→AW	AW
Activity	H,TP	H,TP	H	H,TP	H,TP	H

Figura 4. Politiche di visibilità per i vari OSN.

2.1.4 Analisi Granularità

La granularità dei dati in genere è quella specifica, ciò non vale nel caso del livello di attività in cui è pratica comune accorpare i dati al fine di limitare il volume dei dati generati dal sistema.

Granularity	LinkedIn	Twitter	Orkut	Facebook	MySpace	YouTube
Registration	S	S	S	S	S	S
Networking	S,P	S,P	S	S	S	S
Content	S	S,P	S	S	S	S
Activity	P	P	S	S	S	S

Figura 5. Politiche di granularità per i vari OSN.

La **Persistenza** dei dati solitamente non è contemplata nei *social network*, poiché, ad esempio, le politiche scelte dall'utente non hanno limiti temporali, non cambiano nel tempo a meno che non vengano direttamente modificate dall'utente.

Per quanto riguarda le politiche sulla Privacy, Facebook è l'*OSN* più evoluta, tra quelle attualmente presenti in rete. Tutte le descrizioni del *policy system* di Facebook che seguono sono state ottenute dalla documentazione pubblica dell'*OSN* e da test effettuati personalmente. Il suo sistema di politiche permette all'utente di definire configurazioni particolarmente personalizzate riguardo ai singoli contenuti.

Le possibilità di scelta sulla visibilità di ogni elemento presente sul proprio profilo ricadono principalmente su:

- Amici
- Amici di amici
- Famiglia
- Zona di residenza
- Gruppi localizzati o di lavoro

Gli insiemi di utenti appartenenti a queste categorie sono calcolati dalle informazioni presenti sul profilo personale e sono dinamici. Tuttavia, l'impostazione di una politica per ogni elemento non è esplicita, ma segue il valore globale che l'utente può configurare nel proprio profilo in qualsiasi momento. Tale valore va ad influenzare solamente i post futuri e non quelli precedenti, a cui possono essere riassegnate le politiche solamente in modo esplicito e manuale tramite l'accesso al singolo contenuto. In questo modo, ogni modifica al valore sopracitato influenza il prossimo elemento inserito, permettendo la definizione implicita della politica per ogni condivisione effettuata dallo user.

In caso di *share* (ricondivisione) di un contenuto da parte di un utente che ha i permessi almeno per visualizzarlo, le politiche applicate al nuovo elemento sul profilo dell'utente che ha deciso di ripubblicarlo seguono questo comportamento:

- Il contenuto sarà una copia dell'originale, non verranno comunque trasferiti commenti, like e altri *attachments*.
- La politica avrà un collegamento alla politica originale, quindi ogni modifica alla *policy* da parte dell'utente che per primo ha pubblicato il contenuto sarà subito attiva anche sull'elemento ricondiviso.

- La politica per l'elemento ricondiviso verrà calcolata come intersezione esatta tra la politica dell'utente che ha condiviso l'originale e quella dell'utente che ha deciso di ricondividerla.

Questa configurazione del sistema di gestione delle *policy* porta, ad esempio, a negare l'accesso ad un contenuto ricondiviso da un utente ad un suo amico, nel caso in cui il creatore originale dell'elemento non abbia permesso la visualizzazione ad altri oltre i propri amici. Tuttavia, nel caso in cui l'utente che ha inizialmente creato il contenuto decida di renderlo visibile anche ad altri, anche l'elemento ricondiviso sarà ora visibile al medesimo pubblico. Dall'altra parte, tale sistema non permette di visualizzare commenti e *like* associati al contenuto originale, rendendo di fatto i due elementi totalmente distinti a meno delle politiche.

La scelta effettuata dal team di sviluppo di Facebook è particolarmente recente: in precedenza esistevano alcune falle in casi di conflitti di assegnazione di politiche e tale configurazione li risolve apparentemente per la maggior parte. Rappresenta, quindi, un ottimo punto di partenza tra le proposte disponibili sul web.

I *social network* offrono diversi livelli di privacy, Facebook ad esempio incoraggia i propri utenti a fornire nomi e informazioni personali reali, in altri, invece, la maggior parte delle persone preferisce rimanere anonima.

Nel 2005, è stato condotto uno studio per analizzare 540 profili Facebook di studenti iscritti all'università di Carnegie Mellon. Si è scoperto che l'89% degli utenti aveva fornito dati reali e il 61% utilizzava una propria foto per farsi identificare più facilmente. La maggioranza degli utenti, inoltre, non aveva effettuato modifiche alle impostazioni di privacy, permettendo quindi anche a persone sconosciute di accedere alle proprie informazioni. Le impostazioni di base di Facebook, infatti, permettono l'accesso completo al profilo da parte degli amici, amici degli amici e ai non amici appartenenti alla solita rete. Inoltre, molti utenti non sanno che le impostazioni di privacy vengono resettate, portate allo stato di default, ogni volta che viene aggiornata la *social network*.

È evidente che gli utenti, non essendo pienamente consapevoli dei rischi, possono inserire nelle *OSN* informazioni che potrebbero minare la loro privacy. La politica base relativa alle *policy* di Facebook è stata criticata per la sua troppa permissività.

I problemi di sicurezza e di privacy nelle *social network* derivano dall'enorme quantità di informazioni che ogni giorno viene immessa nelle *social*

network: messaggi, foto, inviti e applicazioni esterne sono spesso un punto d'accesso alle informazioni personali dell'utente.

Nel caso di Facebook, Adrienne Felt, candidata al Ph.D nell'istituto Berkeley, ha esposto una potenziale falla nel *framework*[3], in particolare nella parte riguardante l'*API* delle applicazioni di terze parti.

Tali applicazioni possiedono molte più informazioni personali dell'utente di quelle di cui, in realtà, necessitano, ciò costituisce una possibile violazione dei diritti di privacy dell'utente. Il problema è di tipo implementativo ed è insito nel *framework* di Facebook, ciò significa che, per risolverlo è necessario modificare l'*API*, il che implica anche una modifica delle applicazione che ne fanno uso.

Tuttavia, questo problema pare essere stato risolto recentemente, con l'introduzione della nuova *API* Facebook, completamente diversa da quella precedente e più chiara.

[3]http://www.cs.virginia.edu/felt/privacy/

2.2 Approcci per il Controllo dell'Accesso

La necessità di controllo negli accessi ai contenuti nelle *OSNs* è sempre maggiore, il problema si ripercuote nei sistemi distribuiti basati su architetture *P2P*. Al momento non esiste una soluzione universalmente riconosciuta a questa problematica, che risulta quindi essere motivo di ricerca.

I modelli esistenti di controllo nell'accesso ai contenuti sono i seguenti:

- *Access Matrix Model* (*AMM*)
- *Attribute-based Access Control* (*ABAC*)
- *Role-based Access Control* (*RBAC*)
- *Task-based Access Control* (*TBAC*)
- *Team-based Access Control* (*TMAC*)
- *Spacial Access Control* (*SAC*)

2.2.1 Access Matrix Model

Modello sviluppato da Lampson et al. [26] si basa sull'utilizzo di una **matrice** in cui le righe sono associate ai contenuti e le colonne agli utenti (o viceversa).

In ogni cella, ad esempio, la generica cella alla riga i e colonna j, è presente un valore che rappresenta il livello di accesso e, quindi, indica quali operazioni può effettuare l'utente j sul contenuto i.

Tale modello non è compatibile con le *OSNs*, poiché non è possibile specificare politiche basate sulle caratteristiche del soggetto e del contenuto ed, inoltre, non è adatto ad un ambiente dinamico come quello sociale, in cui modifiche ai diritti di accesso avvengono molto di frequente.

2.2.2 Attribute-based Access Control

Attribute-based Access Control (*ABAC*) è un modello di controllo degli accessi in cui la valutazione viene effettuata in relazione agli attributi dell'oggetto che deve essere acceduto e quelli legati a chi vuole accedervi. Gli attributi sono tutte le caratteristiche definite per le quali è possibile associare un valore, nell'analisi degli accessi possono essere considerati anche gli attributi dell'ambiente.

2.2.3 Role-based Access Control

Sandhu et al. [27] hanno proposto il modello *Role-based Access Control* (*RBAC*) in grado di decidere se permettere l'accesso ai contenuti sulla base del **ruolo** assegnato al soggetto che ne fa richiesta. Il criterio con cui viene assegnato il ruolo può variare a seconda delle caratteristiche degli utenti prese in considerazione. Il ruolo può non essere univoco e ogni risorsa può essere acceduta dall'utente, se il suo ruolo risulta essere abbastanza elevato.

L'uso del modello *RBAC* risulta inadeguato nelle reti sociali, data la sua poca adattabilità alla dinamicità della rete.

Se, per esempio, si vogliono definire i diritti di accesso di un contenuto che può essere acceduto solamente da un utente, sarà necessario definire un nuovo ruolo esclusivo a quell'utente. Se, dopo la modifica di un contenuto, i diritti di accesso cambiano, è necessario cambiare il ruolo associato.

2.2.4 Task-based Access Control

Introdotto da Roshan K. Thomas et al. [28, 29] si basa sul concetto di **task**. L'accesso ad un contenuto viene considerato un *task* e, a seconda di quale contenuto si voglia accedere, i requisiti di accesso cambiano.

È inadeguato in reti sociali, poiché in tali applicazioni non sempre è possibile basare il controllo d'accesso solamente sull'informazione a cui accedere; inoltre, azioni come la revoca o la modifica dei diritti di accesso agli utenti, in questo modello non sono adatte agli ambienti dinamici.

2.2.5 Team-based Access Control

Roshan K. Thomas et al. [30] hanno presentato questo modello che si diversifica dai precedenti, perché prende in considerazione l'accesso ai contenuti da parte di **gruppi**, anzi che da parte di un singolo utente. Formare gruppi risulta più logico di creare ruoli differenti per dividere gli utenti. I gruppi sono formati o in base alle caratteristiche comuni degli utenti, oppure basandosi sui contenuti a cui gli utenti vogliono accedere.

Il modello *Team-based Access Control* (*TMAC*) è basato sul modello *RBAC*, ma al momento non ne esiste un'implementazione completa che, partendo dall'approccio *RBAC*, introduce il concetto di gruppo.

2.2.6 Spatial Access Control

Proposto da Adrian Bullock et al. [31] si prefigge di astrarre i meccanismi di sicurezza agli utenti.

L'architettura del modello è divisa in due moduli: un **modulo di confine**, che divide l'intera rete in piccole regioni in cui l'accesso viene valutato in base al concetto di credenziali e un **grafo di accesso**, che indica le condizioni per effettuare gli accessi e limita i movimenti all'interno dell'ambiente applicativo.

La principale problematica legata a quest'approccio è la difficoltà nel dividere una rete delle dimensioni di una *OSN*. Non fornisce, inoltre, il cosiddetto *fine-grained access control*: non è possibile specificare politiche specifiche e dettagliate.

2.2.7 Tecniche di Access Control per OSN

B. Carminati et al. [32] introducono un approccio diverso che prende il nome di *Rule-based Access Control*.

Le politiche sono definite in base a tre attributi della relazione che lega chi le sottoscrive con il resto degli utenti nella *OSN*: **tipo**, **intensità** e **fiducia**. Quando un utente vuole accedere ad un contenuto deve fornire un certificato che attesti la bontà della relazione che lo lega a chi ha pubblicato l'informazione, dove per bontà di relazione si intende che il tipo, l'intensità e la fiducia siano adeguati alla politica definita per il contenuto. I certificati di relazione sono codificati tramite una chiave simmetrica, conosciuta da un nodo di *storage* e dagli utenti in questione e vengono salvate in un nodo centrale. L'approccio descritto risulta essere flessibile, poiché nessuno può avere accesso ad una risorsa senza avere una connessione con chi la detiene.

La presenza di un nodo centrale rende, però, tale metodo non applicabile in reti sociali distribuite.

J. Domingo-Ferrer et al. [33] estendono il *Rule-based Access Control* introducendo due modifiche: si utilizza un sistema di codifica a chiave pubblica, invece che simmetrica, per la cifratura dei certificati e si rimuove la dipendenza dal nodo centrale, che aveva il compito di mantenere i certificati e calcolare il livello di fiducia. Il livello di fiducia, infatti, viene calcolato, a seguito di una richiesta, direttamente dal possessore del contenuto, basandosi sul certificato ricevuto dal richiedente. Il *framework* modificato è ancora limitato in quanto non permette di definire regole sulle proprietà, statiche e dinamiche,

né degli utenti né delle risorse e non risulta presentare il sufficiente livello di dettaglio negli accessi; non si può specificare, infatti, diritti di accesso in sola lettura o sola scrittura per i singoli contenuti.

La soluzione a queste problematiche si è raggiunta con l'introduzione del linguaggio standard **eXtensible Access Control Markup Language**, in grado di implementare un sistema basato su regole, ma allo stesso tempo in grado di fornire i vantaggi dati dal paradigma *Attribute-based Access Control*. Tale approccio, essendo quello implementato dalla tesi, sarà descritto dettagliatamente nel Capitolo 3.

2.3 Distributed Online Social Network (DOSN)

A causa dei possibili problemi legati ai livelli di servizio non garantibili sopra citati e di interruzione della trust tra l'utente e il provider del servizio, è sorta la necessità di indirizzarsi verso lo sviluppo di **sistemi decentralizzati**.

Nei sistemi decentralizzati non sono presenti server che si occupano della gestione di tutti gli aspetti della *social network*, ma sono gli utenti stessi ad avere funzioni di servizio e di consumatore.

L'approccio architetturale, alla base dei sistemi *DOSN* e, più in generale per le reti *Peer-to-Peer*, che garantisce l'indipendenza da un server centralizzato, può essere classificato in tre diversi stili:

Strutturato: i contenuti sono organizzati in maniera strutturata in una specifica topologia che assicura efficienza in processi specifici, come ad esempio il *routing*.
Si deve però prestare particolare attenzione alla dinamicità del sistema al fine di mantenere la corretta topologia di rete.

Non Strutturato: il sistema non prevede alcuna strutturazione, i contenuti sono distribuiti in base alle loro necessità.
Contrariamente al modello strutturato, alcuni servizi risulteranno poco efficienti, in compenso la dinamicità della rete ha minimo impatto sulle prestazioni del sistema.

Ibrida: tale architettura è un compromesso delle prime due.

Le informazioni legate all'utente sono in possesso di quest'ultimo che decide in che modo e dove vengano salvate.

La gestione della sicurezza e della privacy, quindi, cambia totalmente spostandosi dall'entità centrale ai tanti utenti che utilizzano il sistema. Le problematiche che si incontrano nello sviluppo di una DOSN, oltre a quelle analizzate per i sistemi centralizzati, che dovranno essere gestite diversamente, sono di diverso tipo e legate alla dinamicità della topologia della rete, dovuta all'entrata e uscita degli utenti dal sistema (*churn*).

2.3.1 Data Availability

In primo luogo vi è il problema della **data availability**; il sistema deve garantire che le informazioni di ogni utente rimangano presenti nel sistema, visibili a chi ne ha diritto, anche quando l'utente si scollega dal servizio. Al fine di gestire il problema, vengono utilizzate tecniche come *Distributed Caching* e *Dynamic Data Replication*.

In generale, è necessario replicare i dati per garantirne l'*availability*, stabilire a chi affidare le copie e il numero di repliche da effettuare sono due ulteriori problemi da gestire.

Legato alle problematiche appena descritte vi è il problema dell'analisi del **comportamento temporale** degli utenti. Se un utente effettua la disconnessione dal servizio, il sistema deve affidare i propri dati al peer online che, non solo abbia i diritti di mantenere i dati del nodo uscente, ma che possibilmente rimanga attivo più a lungo. Quindi, necessariamente si deve studiare il comportamento delle sessioni online dell'utente per effettuare stime accurate.

Lo studio della *availability* degli utenti delle *DOSNs*, e dei sistemi P2P in genere, sta acquisendo negli ultimi anni un'importanza sempre crescente. 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 supporto 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*. È stato proposto un modello di **social cache** [39] in cui viene preso in considerazione anche il comportamento temporale dei nodi, in quanto risulta evidente che sia poco utile eleggere come *social cache*

peer che trascorrono un tempo limitato online.

Tra i primi a condurre uno studio empirico sulle caratteristiche delle sessioni e delle interazioni degli utenti in una *OSN* vi sono Golder et al. [9]. Il loro lavoro è basato su dataset estratti da una versione di *Facebook* piuttosto diversa da quella attuale, utilizzata esclusivamente da studenti di college americani; i risultati ottenuti si sono dimostrati un ottimo punto di partenza per la prosecuzione dello studio in questo settore. Gli autori hanno rilevato un'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.). Si è notato, inoltre, come gruppi di studenti facenti parte dello stesso college tendano ad avere comportamenti più simili tra loro rispetto ad altri studenti.

Tali osservazioni, se pur effettuate in un contesto piuttosto diverso rispetto a quello delle *OSNs* attuali, hanno spinto numerosi ricercatori a rivalutare l'importanza del comportamento degli utenti all'interno di un sistema.

In [10] 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.

La traccia dei periodi online di ogni nodo è considerata come un vettore di bit dove l'n-simo bit è pari a 1 se il nodo si trova online durante l'ora n. Viene applicata poi una decomposizione di *Fourier* a ogni segnale; analizzando gli spettri delle frequenze giornaliere e settimanali è possibile individuare comportamenti periodici. Lo studio evidenzia la presenza di due distinti gruppi di utenti: uno gruppo che si trova sempre online, mentre l'altro alterna periodi online e offline, con differenti caratteristiche di periodicità.

L'impatto della disponibilità degli utenti nelle *OSNs* è presentato in [11]. 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; 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 che 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 amici 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.

Anche in altri contesti sono stati proposti sistemi che tenessero conto della disponibilità dei peer della rete. In [12] 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.

2.3.2 Availability Prediction

L'**availability prediction** è stata oggetto di minor interesse negli studi accademici rispetto allo studio generale della disponibilità degli utenti.

In [13] lo scopo degli autori è trovare peer che si comportano secondo un dato pattern di disponibilità. In particolare si ricercano candidati per due diverse tipologie di problemi noti, come **disconnection matching** (un peer cerca un utente che si disconnetterà circa nello stesso momento) e **presence matching** (un peer cerca un utente che sarà online insieme a lui in futuro). Al fine di trovare una soluzione ai problemi descritti è necessario prevedere il comportamento futuro dei peer sulla base della durata delle sessioni passate. Gli autori propongono di usare un semplice predittore binario che stima la presenza online di un peer in una finestra temporale di 10 minuti: la predizione viene fatta basandosi sul 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 19600 peer, ma gli autori ritengono di poter aumentare la cardinalità dell'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 [19] 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* e *Overnet*.

Il comportamento dei peer di *Overnet* risulta sostanzialmente impredicibile, dato lo scarso tempo trascorso online. I peer di *PlanetLab* 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 peer con comportamento periodico, altri caratterizzati da sessioni lunghe e,

infine, gruppi di peer con comportamenti disomogenei.
Si osserva che i peer di *PlanetLab* 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.

2.3.3 Problematiche legate alla Privacy

A livello di privacy, mentre i sistemi centralizzati forniscono all'utente misure d'accesso molto semplici e poco configurabili, le *DOSNs* hanno la possibilità di delegare la definizione di regole precise per la privacy, permettendo direttamente all'utente di definire le policies.

La regolamentazione legata agli accessi ai contenuti definita dagli utenti può venire usata, oltre che a livello di servizio, a livello infrastrutturale per la gestione della migrazione dei profili.

I requisiti delle *DOSN* per proteggere la privacy degli utenti sono:

Autenticazione: si impedisce l'accesso a soggetti non autorizzati, verificando l'unicità dell'identità dell'utente.

Confidenzialità: tale proprietà garantisce che l'accesso ai contenuti di un utente venga effettuato solo in modo autorizzato, sia quando quest'ultimo è connesso al sistema sia quando esso è offline. Al fine di proteggere i dati da accessi non autorizzati, si possono utilizzare diversi approcci, tra cui la crittografia oppure un sistema di controllo degli accessi che applica politiche di privacy.

Integrità: capacità del sistema di impedire modifiche non autorizzate dei contenuti dell'utente; gli utenti devono essere in grado di riconoscere se il contenuto è stato corrotto. Si deve garantire, inoltre, che i dati non vengano intercettati durante una comunicazione.

Availability: quando un utente si disconnette dalla rete le informazioni ad esso correlate devono essere mantenute nel sistema, riassegnandole ad altri utenti o ad una struttura sottostante, come ad esempio una *DHT*.

Anonimicità: abilità del sistema di impedire a soggetti non autorizzati di derivare informazioni personali, azioni o relazioni di amicizia degli utenti.

Fiducia: caratteristica basata sulle interazioni e i comportamenti passati dell'utente, per definire se un certo contenuto è affidabile oppure meno. Grazie a questo tipo di meccanismo risulta più semplice isolare utenti malevoli che rendono il sistema inaffidabile.

2.3.4 Esempi di DOSN

Presentiamo ora le caratteristiche di alcune delle principali DOSNs esistenti:

- **Diaspora**: Diaspora[4] è una *DOSN* con architettura ibrida, caratterizzata da una rete di server decentralizzati (*pod*) che vengono amministrati dai singoli utenti: essi possono in questo modo gestire individualmente i propri dati. Alternativamente gli utenti possono fare riferimento ad un server già esistente, ponendo implicitamente la loro fiducia nel server a cui si aggregano. La comunicazione avviene attraverso dei post, la cui visibilità può essere pubblica o ristretta a uno specifico gruppo di utenti (aspetti) 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. Esiste anche un altro metodo di comunicazione privata fra i nodi della rete che prende il nome di *conversation*.

- **PeerSon**: *PeerSon* [14] è un'infrastruttura ibrida 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 una bacheca, che

[4] https://joindiaspora.com

contiene oggetti e notifiche postati dall'utente stesso o da suoi amici, e offre un servizio di trasferimento di file.

- **Safebook**: *Safebook* [15], anch'esso basato su un sistema *P2P*, è formato da tre componenti principali:

  1. un **Trusted Identity Service (TIS)**, un servizio offline che permette agli utente di acquisire certificati con cui potersi unire alla rete;
  2. le **Matrioska**, strutture concentriche che offrono un servizio di organizzazione degli amici e garantiscono la privacy di ciascun nodo;
  3. un substrato basato su **Kademlia**, che offre il servizio di *look-up*.

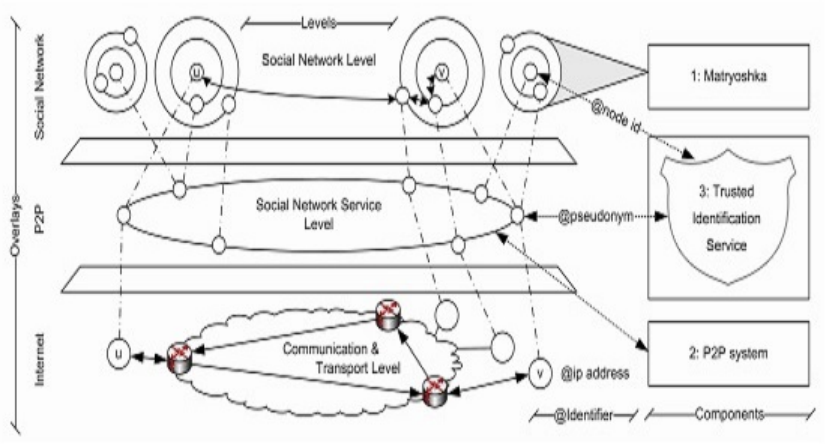


Figura 6. Struttura di SafeBook.

Le componenti sono organizzate in tre diversi livelli: il primo è composto dalla rete generata dalle *matrioska* degli utenti, il secondo che rappresenta l'*overlay P2P* del sistema e il terzo relativo ad internet.
All'interno del sistema si distinguono utenti certificati e utenti anonimi.

- **SuperNova**: *SuperNova* [16] è un sistema non strutturato 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* costituiscono una sorta di indice locale di dimensioni limitate 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* [17] è una piattaforma eseguibile, totalmente distribuita basata su una architettura *P2P* in cui l'overlay sottostante è *FreePastry*. 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. Ogni funzionalità offerta dal servizio è implementata come plugin.

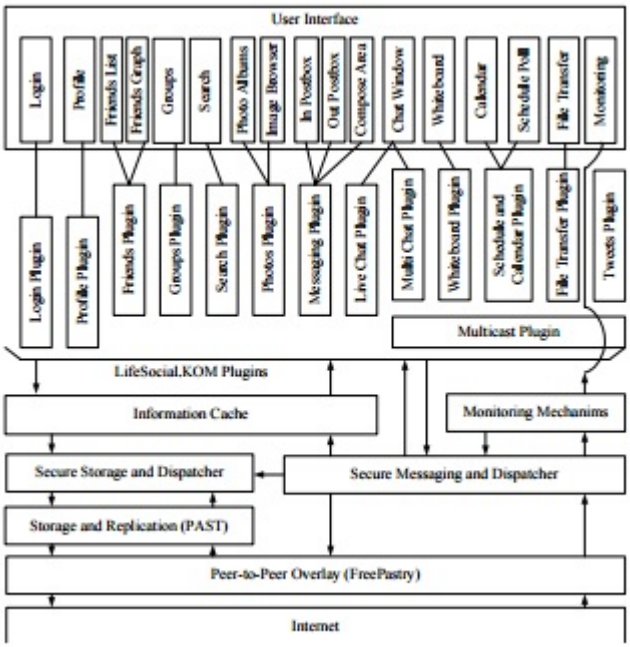


Figura 7. Struttura di LifeSocial.KOM.

- **Confidant**: Citiamo infine *Confidant* [18] 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*.

Analizzeremo ora gli aspetti della privacy nelle *DOSN* presentate in precedenza:

Autenticazione. In Diaspora l'autenticazione viene implementata tramite l'utilizzo del framework *Devise*, in grado di gestire l'identità degli utenti durante il processo di registrazione.
Safebook fa uso del servizio *trusted identification* che assegna ad ogni nodo una chiave di identificazione e il relativo certificato per l'autenticazione.
PeerSoN mette a disposizione delle chiavi pubbliche che permettono agli utenti di autenticare i propri messaggi e garantire la loro integrità; c'è una possibilità che le chiavi vengano revocate per poter essere nuovamente assegnate.
Lifesocial.KOM fornisce ad ogni utente una coppia di chiavi, una pubblica e una privata con le quali gli utenti possono identificarsi univocamente all'interno del sistema.

Access control. Gli utenti di Diaspora organizzano i propri contatti assegnandoli a gruppi che prendono il nome di aspetti; il sistema permette di definire una politica di accesso selezionando per ogni contenuto gli aspetti che possono visualizzarlo.
Per la comunicazione dei dati sono definiti tre diversi livelli di privacy: nessuno, basso e alto. I tre valori si riferiscono al grado di crittografia da utilizzare nella trasmissione e nell'immagazzinamento dei dati; nel primo caso avviene tutto in chiaro, nel secondo la crittografia viene utilizzata solamente su richiesta dell'utente. Nell'ultimo caso ogni utente possiede una coppia di chiavi, una privata e l'altra pubblica, con cui applicare la crittografia.
Nel terzo caso solo il traffico fra server e server viene codificato, il resto delle comunicazioni avvengono in chiaro.
In *Safebook* i contenuti dell'utente possono essere privati pubblici o

protetti. In caso le informazioni siano private non possono essere visualizzate da altri utenti, se sono protette vengono codificate, altrimenti vengono memorizzate e trasmesse in chiaro.
I messaggi vengono crittografati utilizzando la chiave pubblica di chi riceve il contenuto.
PeerSoN permette agli utenti di definire delle semplici politiche di accesso in modo da specificare chi può visualizzare una certa informazione. I dati sono codificati utilizzando la chiave pubblica di chi può accedervi.
LifeSocial.KOM fornisce la possibilità agli utenti di definire liste di accesso, *Access Control Lists (ACLs)*, composte dagli utenti con i permessi per visualizzare delle informazioni.
Tutte le comunicazioni sono effettuate mediante crittografia, in particolare ogni dato viene codificato tramite una nuova chiave simmetrica che viene apposta al dato dopo essere stata codificata utilizzando gli le chiavi asimmetriche degli utenti in grado di decrittare l'informazione.

Integrità. In Diaspora i dati dell'utente sono riconoscibili durante le comunicazioni, tramite l'inserimento delle chiavi pubbliche e private all'interno dei contenuti.
Allo stesso modo anche in *Safebook* ogni dato è firmato da chi lo pubblica, durante le comunicazioni l'integrità è garantita sia nel caso di trasmissioni dirette, inserendo l'identificativo del ricevente nel contenuto, che nel caso di trasmissioni indirette utilizzando le chiavi dei nodi attraversati dall'informazione.
PeerSoN e *Lifesocial.KOM* garantiscono l'integrità dei contenuti attraverso l'inserimento delle chiavi pubbliche nell'informazione trasmessa.

Availability. Diaspora, come già descritto, ha una struttura ibrida, ogni utente può essere collegato ad un server, che prende il nome di *pod*, oppure agire come tale.
La gestione delle informazioni, nel caso di uscita/entrata degli utenti è delegata quindi ai *pod* similmente a come avviene nei sistemi centralizzati.
In *Safebook* ad ogni utente è associata la *matrioska*, una struttura ad anelli concentrici composti dai nodi amici. Al centro della *matrioska* è presente il core che, nel primo livello del sistema, è associato all'identificativo dell'utente; la forza della relazione fra i nodi e l'utente è

direttamente proporzionale alla distanza dall'anello, in cui è posizionato il nodo, al core.
Si individuano perciò due ruoli specifici: *mirror* ed *entrypoint*; i mirror rappresentano i nodi più "fidati" dell'utente (i più vicini al core).

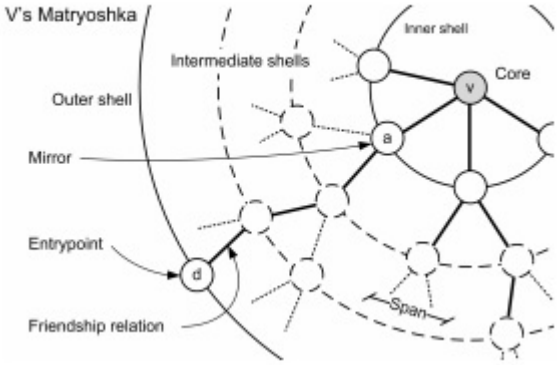


Figura 8. Struttura del core di SafeBook.

Gli *entrypoint* hanno la funzione di *gateway*, ovvero si occupano di instradare correttamente le richieste indirizzate all'utente.
La gestione dell'availability è assegnata ai *mirror* che contengono tutti i contenuti del core codificati.
PeerSoN utilizza i server di look-up per mantenere i contenuti degli utenti disconnessi dal sistema. Le informazioni vengono codificate e replicate in modo da averle sempre a disposizione.
Lifesocial.KOM garantisce l'availability dei dati tramite la *DHT* (*FreePastry*), in cui i contenuti vengono replicati.

Anonimicità. Al fine di garantire l'anonimicità, l'identità degli utenti e le relazioni che li legano non devono essere dedotte dalle DOSNs. *Safebook* provvede a mantenere nascoste, a chi non ha il diritto di accedervi, le informazioni personali degli utenti tramite il *Trusted Identification Service* che garantisce un identificativo univoco per ogni livello del sistema.
L'identità di un utente è conosciuta solo dai peer di cui si fida, che sono gli unici a poter associare l'indirizzo ip con l'identificativo dell'utente.

Gli altri sistemi non forniscono meccanismi specifici atti a garantire l'anonimicità.

Fiducia. L'unica *DOSN*, tra quelle descritte, che prevede un sistema di fiducia è *SafeBook*, il quale utilizza le *Matrioska*, anelli concentrici composti esclusivamente dai nodi di cui l'utente ha fiducia.

2.4 Reti Complesse

L'analisi delle reti sociali si basa sull'utilizzo di tecniche derivanti dalla teoria dei Grafi.

La rete sociale è rappresentata da un **grafo**, dove i vertici corrispondono agli **utenti**, mentre gli archi che li collegano indicano le **relazioni sociali esistenti tra gli utenti** (ad esempio di amicizia o lavoro). Nel passato sono stati effettuati studi al fine di comprendere le proprietà dei singoli nodi o dei singoli archi di un grafo, ma l'incremento delle dimensioni e della complessità delle reti ha reso poco significativi i risultati precedentemente ottenuti. Di conseguenza, al fine di ottenere una visione "globale" della rete, è stato necessario cambiare approccio sviluppando e formalizzando nuove metriche e misure.

Le reti moderne sono composte da milioni di nodi e archi e vengono in genere riferite come **Complex Network**; le OSNs sono un esempio di questa tipologia di reti.

Di seguito descriviamo alcune proprietà rilevanti delle reti complesse che saranno utili nella definizione del framework proposto in questa tesi.

2.4.1 Small World

Nel 1960 il sociologo americano Stanley Milgram [20] condusse un famoso esperimento: selezionò 200 persone casualmente negli stati del Nebraska e del Kansas e chiese loro di consegnare una lettera ad un broker che viveva a Boston, conoscendo solamente il suo nome. I partecipanti all'esperimento inoltravano la lettera solo alla persona conosciuta che reputavano essere più vicina a conoscere il destinatario.

L'esperimento è conosciuto con il nome "*six degree separation*", poiché è risultato che il numero massimo di passaggi di lettera necessari per portare l'informazione dal Nebraska o Kansas al broker a Boston è sei.

Il risultato ottenuto testimonia che, anche in reti complesse - caratterizzate da un numero considerevole di nodi ed archi - si ha un diametro di rete[5] piccolo rispetto alle dimensioni complessive. La proprietà descritta prende il nome di **Small World** ed è strettamente legata alla velocità con cui le informazioni vengono diffuse nella rete [22].

Anche nelle reti sociali è nota la validità di questa proprietà, come esempio specifico possiamo citare uno studio che ha dimostrato come il diametro della rete di *Facebook* si stia restringendo [21]. Nel 2008, in media, il cammino minimo passante tra due generici nodi era lungo 5,28 hop[6], valore che nel 2011 è risultato di 4,74.

2.4.2 Clustering

Il coefficiente di **clustering** è utilizzato per verificare la presenza di gruppi di nodi, comunità, che prendono il nome di **cluster**. La caratteristica principale dei *cluster* è che ogni nodo che li compone è strettamente legato agli altri componenti, mentre è debolmente relazionato con il resto della rete.

Il coefficiente di *clustering* può essere valutato localmente per uno specifico nodo oppure globalmente.

Il primo approccio fornisce una misura sulla quantità di archi esistenti fra i vicini di un nodo n in rapporto al numero massimo teorico di archi presenti fra loro.

Il secondo certifica la qualità del partizionamento in comunità dei nodi della rete: può essere espresso come la media di tutti i coefficienti di *clustering* locali ed è un valore compreso nell'intervallo $[0, 1]$. Un valore vicino allo zero indica che sono presenti molti archi che connettono nodi distanti fra loro, al contrario un valore vicino a uno denota una rete con molti archi fra nodi vicini.

2.4.3 Misure per l'Analisi di Reti Complesse

Una delle proprietà importanti per studiare la struttura di una rete riguarda il *grado* dei nodi. Se in media i nodi hanno un numero di archi uscenti simile, la rete avrà una struttura regolare, in caso contrario saranno presenti nodi "chiave" da cui partono un numero elevato di link e nodi con pochi collegamenti.

[5] Il diametro di rete è definito come la massima distanza fra due nodi della rete.
[6] spostamento tra due nodi direttamente connessi.

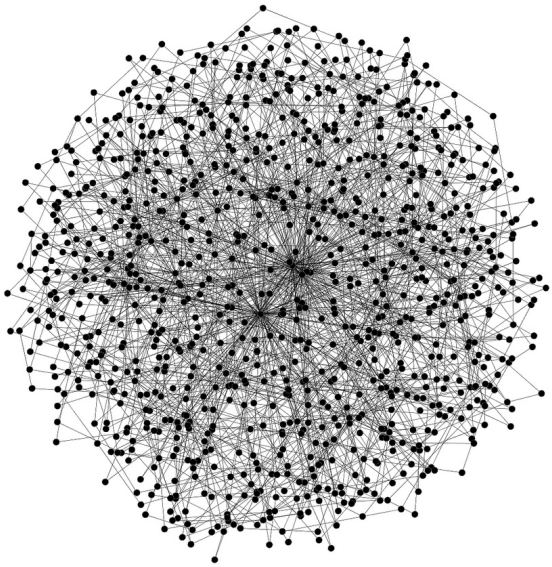


Figura 9. Esempio di rete densa.

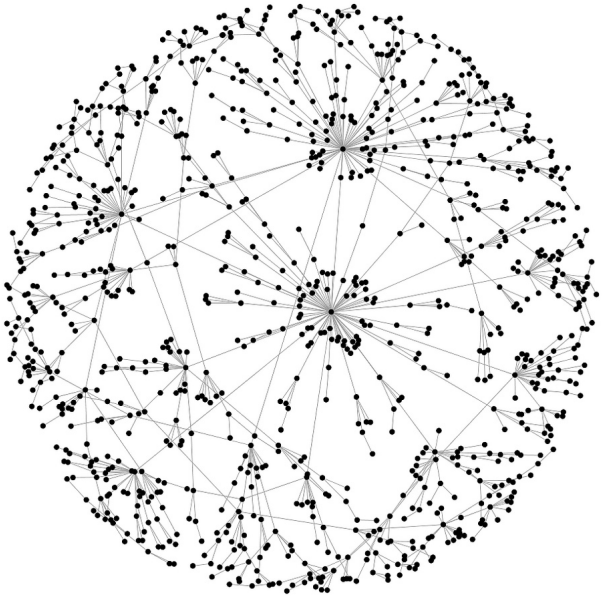


Figura 10. Esempio di rete sparsa.

Individuare i nodi chiave è importante, poiché possono essere utilizzati come punto di partenza per diffondere informazioni, all'interno della rete, nel modo più efficiente possibile (oppure come *social cache*).

Rimuovere un nodo chiave può portare gravi danni alla struttura di rete, partizionando il grafo originale in componenti separate.

In termini strutturali, ciò significa che gli archi non sono distribuiti uniformemente e che il grafo è composto da pochi nodi con valore alto della distribuzione e molti con valore basso.

Reti che presentano una struttura simile sono dette "**Scale-free Network**", in cui la distribuzione del grado degli archi uscenti dei nodi è indipendente dalla dimensione della rete.
Interessante è anche capire in che modo i nodi sono collegati fra loro: i nodi con alto grado di archi uscenti possono essere collegati con nodi simili oppure con nodi aventi un basso grado.

Al fine di misurare il grado di correlazione fra i vari nodi si deve analizzare la correlazione lineare tra due variabili x e y utilizzando la **Pearson's Correlation Index**, illustrato di seguito.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

dove $\bar{x}$ e $\bar{y}$ sono i valori medi delle variabili x_i e y_i.

L'indice fornisce importanti informazioni relative alla solidità della struttura del grafo; se il valore di correlazione è alto, i nodi con molti archi uscenti sono collegati con nodi simili e la struttura risulterà solida; il fallimento di un nodo non comporta particolari problemi. Al contrario se l'indice è vicino a zero, la rete risulterà sparsa e il fallimento di un nodo chiave può provocare, come già descritto, danni seri alla struttura.

La misura del"indice di Pearson fornisce, quindi, un'indicazione sulla *resilience*: la capacità della rete di mantenere il servizio ad un buon livello quando si verificano fallimenti o rimozioni di nodi. La valutazione di questa proprietà può avvenire effettuando la rimozione di nodi in maniera casuale o specifica.

L'indice di centralità è una metrica importante per descrivere le proprietà delle reti. Tale misura è volta allo studio dell'importanza di un nodo all'interno della rete.

Diversi indici sono utilizzati per il calcolo della centralità, i più comuni sono i seguenti.

Il **Degree Centrality (DC)** misura la connettività di un nodo alla rete in termini di archi entranti; è definito come:

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

La **Closeness Centrality (CC)** è la misura della distanza media fra un nodo e ogni altro; fornisce un'indicazione della velocità di propagazione delle informazioni a partire dal nodo scelto verso il resto della rete.
L'equazione che la definisce è la seguente, tenendo conto che $d(v, t)$ è il cammino di distanza minima fra il nodo v e il nodo t.

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

Una terza misura di centralità è la **Betweenness Centrality**: questa valuta l'importanza di un nodo nella rete basandosi sul flusso di informazioni che lo attraversano. La *Betweenness Centrality* del nodo v misura quanti, fra i cammini minimi tra un nodo di partenza s e un nodo di arrivo t, per ogni possibile coppia s,t, sono passanti per il nodo v.

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

dove $\sigma_{s,t}$ è il numero di cammini minimi dal nodo s al nodo t.
Tale misura ci fornisce un valore 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.

2.5 Ego Networks

Un concetto interessante riguardante le *OSN*, introdotto di recente, è il modello **Ego Networks** (EN).
Consiste nella rappresentazione di un utente, che prende il nome di **ego**, delle sue amicizie dirette, chiamati **alters**, e dei loro legami sociali in un grafo.

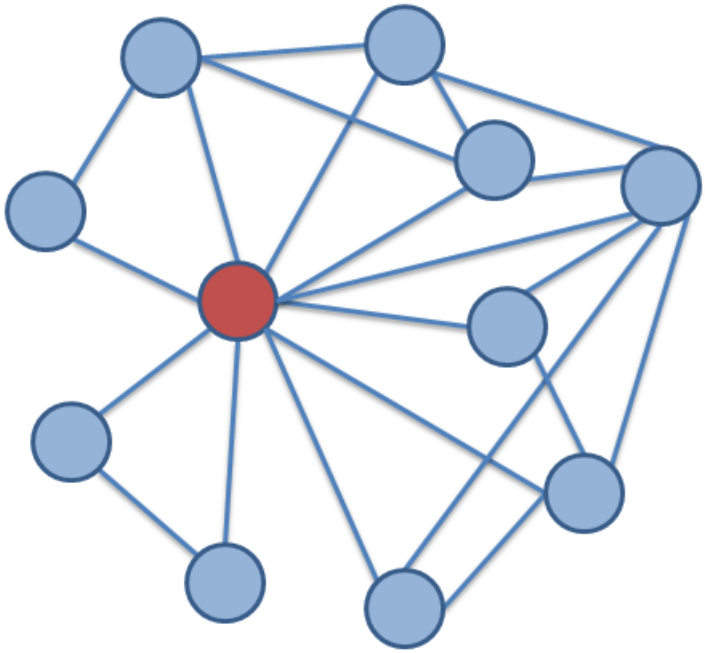


Figura 11. Rappresentazione di EG, dove il nodo rosso è l'Ego.

2.5.1 Dunbar

L'**ego network** di un utente può essere arbitrariamente grande, molti studi sociologici e antropologici hanno evidenziato, però, come mantenere i rapporti sociali sia dispendioso dal punto di vista delle capacità cognitive di una

persona. Nello specifico, si è individuato che il limite massimo alle relazioni mantenibili da un *ego network* è di circa 150 amicizie, il cosiddetto numero di **Dunbar**.

Altri studi, impegnati nell'analisi dei social network, hanno mostrato come sia possibile, nella rete, riconoscere delle cerchie intorno all'*ego*, che rappresentano i diversi tipi di rapporti sociali e la relativa forza; avvicinandosi all'*ego* la solidità dei legami aumenta.
La forza del rapporto sociale è rappresentata da un valore numerico determinato come la combinazione lineare della durata, dell'intensità emozionale, della mutua confidenza e dei servizi utilizzati in comune, che caratterizzano il legame.

Una possibile definizione delle cerchie di *Dunbar* può essere fatta considerando le diverse frequenze di contatto fra gli *ego* e gli *alter*.

Il cerchio più vicino all'*ego* è chiamato "**Support clique**" ed è composto in media da 5 *alter* e descrive il gruppo di persone a cui l'*ego* si potrebbe rivolgere in caso di problemi finanziari o di stress emotivi. La tipica frequenza di contatto stimata è almeno di un'interazione a settimana.

La seconda cerchia è definita "**Sympathy group**", composta in genere da 15 membri e descrive gli individui contattati dall'*ego* almeno una volta al mese.

La cerchia successiva è il cosiddetto "**Affinity group**", 50 membri, mentre l'ultima, che può arrivare fino a 150 *alters*, è definita come "**Active Network**". I legami sociali con altri utenti non facenti parte delle cerchie descritte sono da considerarsi come semplici conoscenze.

Le cerchie esterne contengono le cerchie interne e l'*Active Network* rappresenta la porzione di rete con cui l'utente ha rapporti sociali significativi.
Una rappresentazione delle cerchie di Dunbar è mostrato in Figura 12.

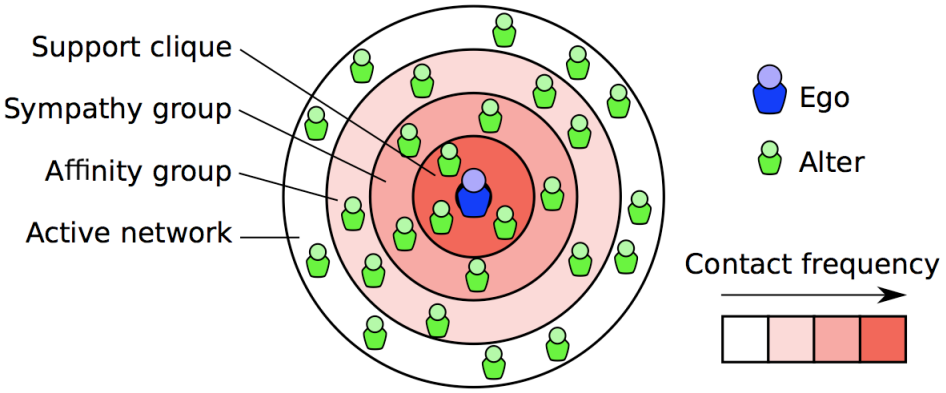


Figura 12. Rappresentazione dei cerchi di Dunbar.

2.6 Il Protocollo Pastry

2.6.1 Introduzione alle DHT

Le **Distribute Hash Table**, *DHT*, sono un modo per implementare un overlay strutturato di reti nei sistemi peer to peer. Due aspetti importanti da sviluppare in questo tipo di applicazioni, sono l'identificazione di nodi e dati nella rete e in che modo si devono memorizzare le risorse all'interno della stessa.

La risoluzione di questi due problemi, deve tenere conto della scalabilità e della robustezza del sistema.
La **scalabilità** rappresenta la quantità di traffico di dati e la memoria utilizzata da ogni nodo in rapporto al numero totale di nodi presenti nella rete. La **robustezza**, invece, è la proprietà del sistema ad adattarsi ai cambiamenti dovuti agli ingressi e alle uscite dei nodi, talvolta non previste.

Due sono stati gli approcci per la gestione di queste tematiche: le reti **non strutturate** e quelle **strutturate**.

Nel primo caso non c'è un concetto di indirizzamento e si ricercano le risorse utilizzando un attributo del dato da trovare, il quale può non essere univoco, propagando l'informazione a tutti i vicini, che la invieranno nuovamente ai propri e così via, finché la risorsa verrà trovata. Ovviamente questa soluzione non è scalabile, a causa dei colli di bottiglia che si possono creare data la mancanza di struttura della rete, anche se si può ridurre notevolmente il traffico utilizzando diverse ottimizzazioni. Inoltre, vi è la possibilità, nelle

ricerche, di avere falsi negativi. Non essendoci alcuna regola nella costruzione della struttura, i cambiamenti e il crescere del numero di elementi nella rete non comportano nessuno sforzo di adattamento.

Le *DHT*, sono classificate come reti strutturate.
Il sistema è affidabile in caso di cambiamenti ed è scalabile nonostante una grande quantità di nodi gestiti.

La prima cosa da fare per implementare una *DHT* è definire uno spazio logico, che dev'essere abbastanza grande per evitare collisioni fra peer (falsi positivi); ad utenti e risorse vengono attribuiti un indirizzo logico univoco utilizzando una funzione *Hash* uniforme, tale indirizzo ha una collocazione nello spazio logico. L'idea generale è quella di associare ad ogni nodo un insieme di indirizzi contigui corrispondenti a delle risorse. Se si vuole avere replicazione parte di questo insieme può essere condiviso da più peer.

Solitamente il routing avviene cercando, tramite l'*overlay network*, l'indirizzo logico della risorsa; ogni nodo ha una visione parziale della rete di cui tiene traccia nella *routing table*. Il nodo che effettua l'operazione di ricerca, solitamente, la propaga al nodo conosciuto più vicino, a chi gestisce lo spazio di indirizzamento logico in cui ricade l'indirizzo da trovare.

Al fine di inserire un nuovo nodo, si utilizza un server di *bootstrap* che permette al peer di interfacciarsi col sistema la prima volta. Il peer applica, tipicamente sul suo indirizzo ip, la funzione *hash* che restituisce la chiave k. Si procede dunque ad una ricerca di k nella rete, si trova così la collocazione logica del nuovo peer, che dovrà prendersi carico di una parte del lavoro gestita dal peer vicino. Il costo del traffico dei messaggi fra i nodi è stimato essere di $O(logN)$, esattamente come la memoria occupata dalle *routing table* dei singoli nodi.

L'uscita volontaria dalla rete di un peer viene gestita dividendo il carico di dati ai suoi vicini, cancellando la sua *routing table* e aggiornando quella dei nodi nelle sue vicinanze.

Allo scopo di prevenire la perdita delle informazioni, oltre alla replicazione, si usano meccanismi di *refresh*: ogni determinato lasso di tempo, ogni nodo testa l'operatività dei vicini.
Ci si accorgerà quindi se si verifica un fallimento di un peer e sarà necessario semplicemente aggiornare le *routing table* dei nodi nelle vicinanze di quello fallito.

2.6.2 Pastry

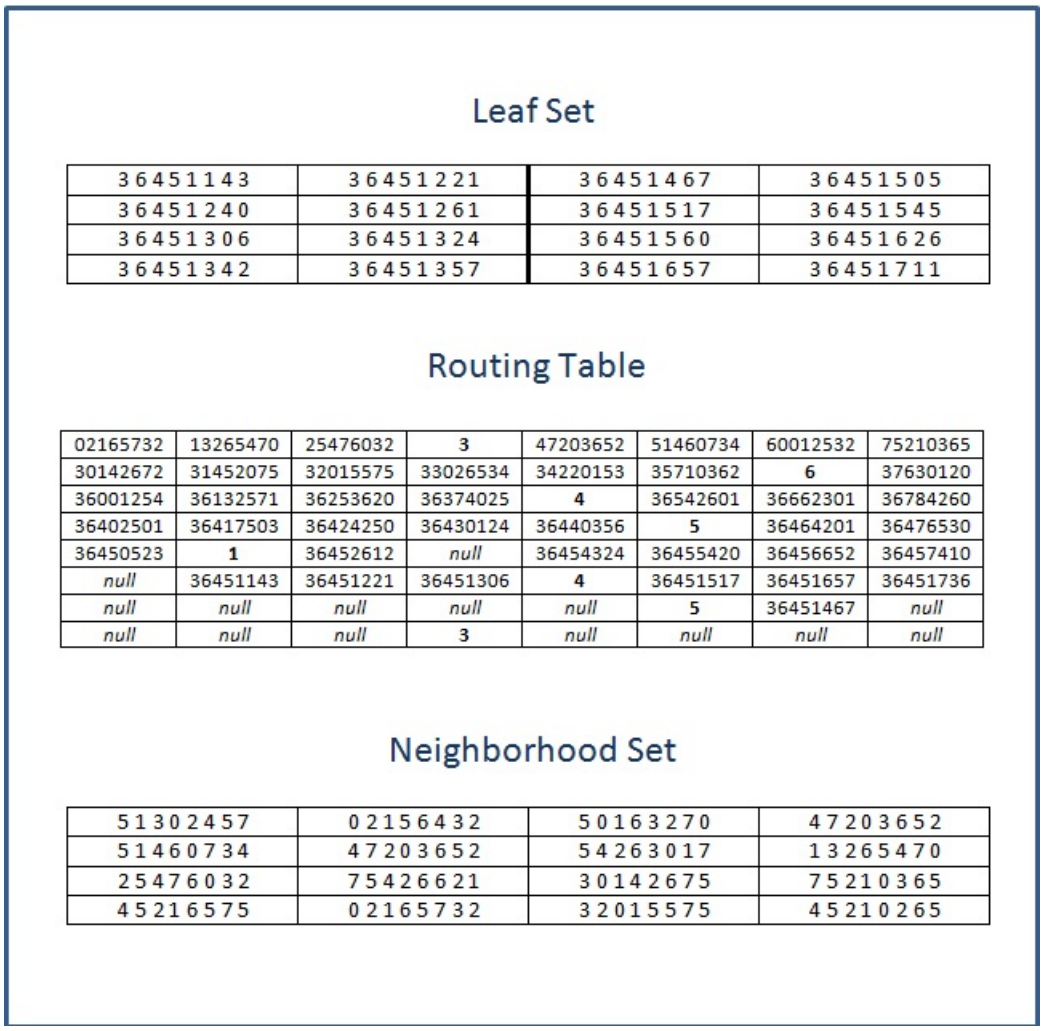
Il protocollo *Pastry* è una specifica versione di *Distributed Hash Table*, che è stata definita da A. Rowstron (Microsoft Research) e P. Druschel (Rice University) [36]. Si tratta di un protocollo completamente decentralizzato, che non fa quindi uso né di servers né di super-peers.
Inoltre, tiene conto della località durante la procedura di *routing* ricorsivo, grazie alla metrica di prossimità; riduce, quindi, la latenza data dalla distribuzione delle risorse, la quale è messa in atto dalla funzione *hash* uniforme, che risulta essere totalmente casuale e non connessa in alcun modo alla località.

Pastry si basa sui **Plaxton Mesh**[7] e **Prefix Matching**[8], come anche *Tapestry* e *Kademlia*.
L'identificatore (chiamato *nodeId*) è generalmente a 128 bit, generato autonomamente dal nodo che sta entrando nella rete; il criterio su cui fare *hashing* è a scelta dell'implementatore (solitamente SHA-1 su IP).

Nell'esempio presentato di seguito si sono usati identificatori a 24 bit ($8digits * b$) con base 8 (2^b), adatti ad una configurazione di rete con 10^7 nodi (16.700.000).

Ogni nodo, affinché la struttura della rete possa essere rispettata e mantenuta, preserva alcune informazioni riguardo agli altri nodi.
Queste informazioni, cioè i nodeIds di questi ultimi, sono tenute in tre tabelle all'interno dello stato del nodo stesso: **Routing Table**, **Leaf Set**, **Neighborhood Set**.

[7]Struttura dati distribuita con particolari proprietà.
[8]Comparazione dei prefissi di stringhe.



Leaf Set

36451143	36451221	36451467	36451505
36451240	36451261	36451517	36451545
36451306	36451324	36451560	36451626
36451342	36451357	36451657	36451711

Routing Table

02165732	13265470	25476032	**3**	47203652	51460734	60012532	75210365
30142672	31452075	32015575	33026534	34220153	35710362	**6**	37630120
36001254	36132571	36253620	36374025	**4**	36542601	36662301	36784260
36402501	36417503	36424250	36430124	36440356	**5**	36464201	36476530
36450523	**1**	36452612	*null*	36454324	36455420	36456652	36457410
null	36451143	36451221	36451306	**4**	36451517	36451657	36451736
null	*null*	*null*	*null*	*null*	**5**	36451467	*null*
null	*null*	*null*	**3**	*null*	*null*	*null*	*null*

Neighborhood Set

51302457	02156432	50163270	47203652
51460734	47203652	54263017	13265470
25476032	75426621	30142675	75210365
45216575	02165732	32015575	45210265

Figura 13. Esempio di tabelle di Pastry.

Routing Table La *Routing Table* è composta da $\lceil log_{(2^b)}N \rceil$ righe (cioè la lunghezza degli ID dei nodi, dipendente dalla configurazione del Network) ognuna di $2^b - 1$ elementi (nell'esempio, essendo $b = 3$, avremmo 7 entry per riga); l'entry esclusa è quella, alla riga j, nella quale il simbolo j-esimo corrisponde al simbolo alla posizione j del *nodeId* corrente. Quindi, ad esempio, avemmo che, alla riga 2, la cella che al terzo simbolo dovrebbe avere un 4 (terzo simbolo del *nodeId*) resta vuota.
Tutte le altre entry $R_{j,i}$ della riga j sono riempite con l'indirizzo IP di un nodo della rete che ha i simboli $[1...j-1]$ uguali a quelli di *nodeId* e il j-esimo uguale all'indice della colonna i; ad esempio, $R_{4,4}$ dovrà avere i simboli [1...3] uguali a quelli di *nodeId* e il simbolo di indice 4 uguale a 4 (cioè i).
Tutti i nodi inseriti nella tabella sono scelti mediante la metrica di località utilizzata, quindi è possibile notare nodi presenti nel *Neighborhood Set*, illustrato in seguito; un esempio è l'entry $R_{1,0}$ e quasi tutte le entry alla prima riga.
E' possibile avere entry lasciate a *null* nel caso in cui non vi siano nodi nella rete corrispondenti numericamente ai criteri per quella cella.

Leaf Set Il *Leaf Set* ha cardinalità pari a $|L|$, che nell'esempio è stata scelta uguale a $16(2 * 2^b)$.
E' suddiviso in due parti: la parte sinistra contiene solamente nodi con identificatore numericamente inferiore al *nodeId* corrente, mentre la parte destra è riempita con nodi identificati da numeri maggiori del *nodeId* corrente.
Questi nodi sono scelti solamente mediante la prossimità numerica e non quella metrica.
Nell'esempio avremmo nella parte sinistra gli 8 nodi numericamente più prossimi a *nodeId* per valori inferiori e nella parte destra gli 8 numericamente più prossimi per valori superiori.

Neighborhood Set Il Set dei vicini del nodo attuale, solitamente, ha la stessa cardinalità $|M|$ del *Leaf Set* (in questo caso 16).
Esso contiene tutti quei nodi che sono stati individuati come i più vicini, secondo la metrica di località, rispetto al nodo stesso.
Durante la fase di *routing* questa tabella non è utilizzata, se non in rari casi ed è atta principalmente al mantenimento delle proprietà di località di *Pastry*.

La procedura di **Routing** di una query sarà presentata, per passi, riguardo all'esempio di pagina 49.

```
(1)  if (L_{-⌊|L|/2⌋} ≤ D ≤ L_{⌊|L|/2⌋}) {
(2)      // D is within range of our leaf set
(3)      forward to L_i, s.th. |D − L_i| is minimal;
(4)  } else {
(5)      // use the routing table
(6)      Let l = shl(D, A);
(7)      if (R_l^{D_l} ≠ null) {
(8)          forward to R_l^{D_l};
(9)      }
(10)     else {
(11)         // rare case
(12)         forward to T ∈ L ∪ R ∪ M, s.th.
(13)             shl(T, D) ≥ l,
(14)             |T − D| < |A − D|
(15)     }
(16) }
```

Figura 14. Algoritmo di routing di Pastry.

1. Viene controllato, per prima cosa, se la chiave ricade all'interno dell'intervallo del *Leaf Set* (quindi si controlla solo sulla prima e sull'ultima entry di questo); in tal caso, l'oggetto ricercato è assegnato a una delle "foglie" del nodo e può essere individuato direttamente.
   Se, nell'esempio, la chiave ricercata fosse 36451236, un solo passo di *routing* dal nodo corrente a 36451240 sarebbe sufficiente per terminare la ricerca.

2. Nel caso in cui la prima fase abbia dato esisto negativo, si procede all'utilizzo della *Routing Table*; sarà considerata solamente la riga successiva a quella contenente nodi in cui i simboli in comune tra chiave e nodo corrente sono equivalenti, i. e. si cerca di passare il *routing* ad un nodo con un digit in più condiviso con la chiave.
   Se, nell'esempio, la chiave cercata fosse 32510574 si passerebbe l'instradamento della *query* al nodo 32015575 ($R_{1,2}$), in quanto condivide in più con la chiave il digit in posizione 1 (il 2) che il nodo corrente non condivide.

Questa procedura garantisce la convergenza verso il nodo cui è assegnato l'oggetto ricercato, ma tale entry potrebbe essere stata impostata a *null*, come nel caso in cui si ricercasse 36453205: il nodo destinazione del *routing* sarebbe quello contenuto nell'entry $R_{4,3}$, la quale non contiene alcun indirizzo IP.

Figura 15. Esempio di un possibile routing, agli ultimi 4 hops, di una query per l'oggetto 52647061, gestito da 52647059. L'ultimo hop è individuato nel Leaf Set.

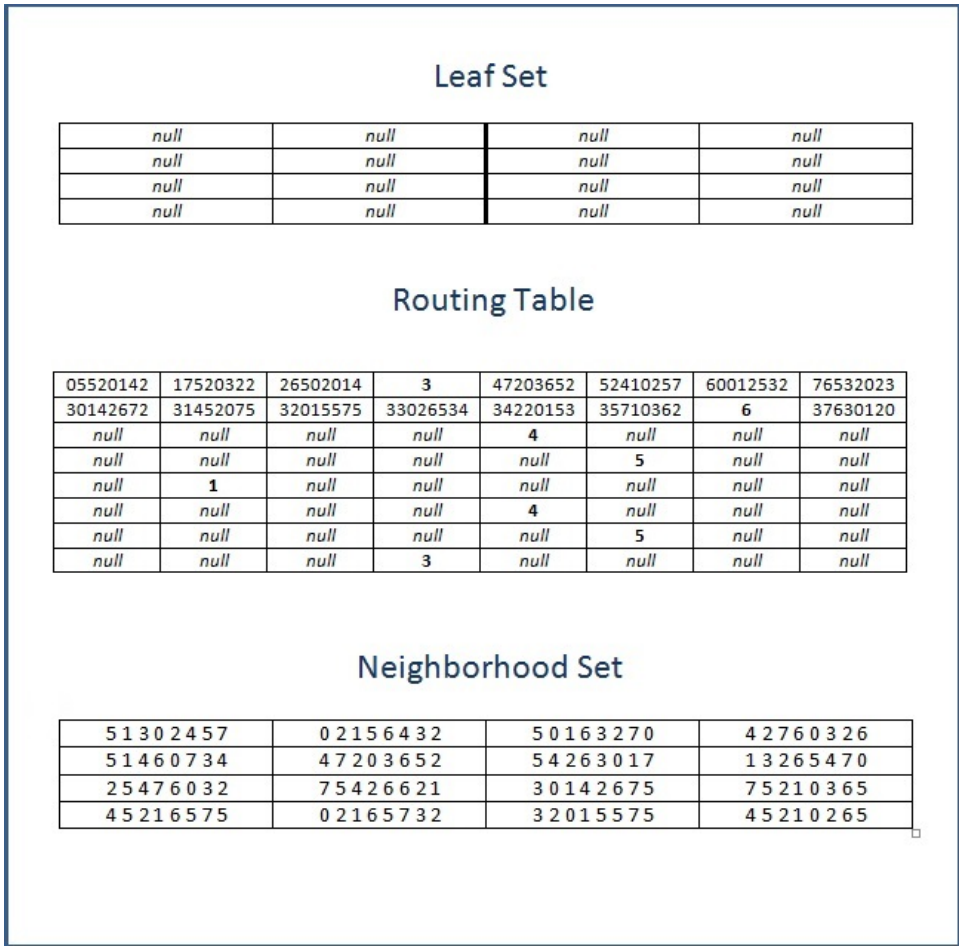
3. Nel caso in cui accada il fatto appena descritto, si ricerca un nodo che condivida con la chiave un prefisso di simboli lungo almeno quanto quello comune tra chiave e nodo corrente sia nella *Routing Table* (solo la stessa riga considerata alla fase 2), sia nel *Leaf* sia nel *Neighborhood Set* e la cui distanza numerica sia minima.
   Nell'esempio precedente si sarebbe selezionato il nodo appartenente alla cella adiacente 36452612, ma se nel *Neighborhood Set* fosse stato contenuto un nodo 36452872, ad esempio, sarebbe stato selezionato quest'ultimo, giacché più vicino numericamente alla chiave (oltre ad avere meno latenza, ma è un caso raro).
   È facilmente dimostrabile che tale passo contribuisca alla convergenza, poiché il nodo individuato sarà numericamente più vicino alla chiave in ogni caso.

Nel complesso, la procedura di *routing* è dimostrata impiegare, al caso medio, $\lceil log_{(2^b)}N \rceil$ passi, con una probabilità trascurabile che anche il terzo passo fallisca, aumentando il numero di *hops* richiesti di 1. Il caso pessimo, dovuto a fallimenti simultanei e conseguente cattiva configurazione dell'*Overlay* (molto improbabile), ha costo $O(N)$.

Mantenimento e Adattabilità

In una normale rete *P2P* il *churn* è elevato e occorre gestire efficientemente l'**entrata**, l'**uscita** e il **fallimento dei nodi**.

Entrata di un nuovo nodo. Quando un nuovo nodo ha intenzione di unirsi alla rete, deve venire a conoscenza (o, alternativamente, venire informato) dell'indirizzo IP di un nodo a lui vicino in termini di località (potenzialmente un nodo del suo *Neighborhood Set*).
Assumiamo che il nuovo nodo sia esattamente quello presentato come esempio a pagina 49 e che venga notificato da un *Bootstrap Server* della presenza del nodo A 47203652, a lui prossimo geograficamente; inizialmente avremmo il nodo corrente con uno stato interno vuoto.
Il primo *Set* a poter essere inizializzato è quello dei vicini, in quanto, per configurazione, A e i suoi vicini saranno i più prossimi al nodo entrante.
A questo punto A si preoccuperà di ricercare 36451453, cioè il nodo appena entrato, che non è ancora presente nello stato interno di nessun nodo della rete; in realtà, il messaggio di *routing* è contrassegnato dal tipo "*join*" e porta nel suo *header* l'indirizzo del nuovo nodo: questo per permettere a tutti i nodi incontrati nella fase ricorsiva di *routing* di inviare a quest'ultimo informazioni utili alla sua inizializzazione.
Il primo nodo ad inviare informazioni al nodo entrante sarà A stesso: *Neighborhood Set* completo, come già detto, e la riga 0 della *Routing Table*, in quanto non vi sono vincoli sul prefisso comune a tale riga; in seguito il primo contattato da A, che chiameremo B (ad esempio, 35710362) procederà ad inviare la riga 1, cioè quella con un digit comune al nuovo nodo e così via.
Ovviamente, ogni riga ricevuta potrebbe avere informazioni in più e/o in meno rispetto a quelle necessarie (ad esempio, la riga 1 non avrà l'entry 4 e la 2 non avrà l'entry 5, dato che i rispettivi nodi da cui sono state ricevuti non le necessitavano) e verranno completate con il nodo stesso che ha inviato la reply.



Leaf Set

null	null	null	null
null	null	null	null
null	null	null	null
null	null	null	null

Routing Table

05520142	17520322	26502014	3	47203652	52410257	60012532	76532023
30142672	31452075	32015575	33026534	34220153	35710362	6	37630120
null	null	null	null	4	null	null	null
null	null	null	null	null	5	null	null
null	1	null	null	null	null	null	null
null	null	null	null	4	null	null	null
null	null	null	null	null	5	null	null
null	null	null	3	null	null	null	null

Neighborhood Set

51302457	02156432	50163270	42760326
51460734	47203652	54263017	13265470
25476032	75426621	30142675	75210365
45216575	02165732	32015575	45210265

Figura 16. Stato interno del nodo entrante 36451453 dopo aver ricevuto reply dai nodi A e B, con rispettivamente (Neighborhood Set, riga 0 Routing Table) e riga 1 Routing Table. $R_{0,4}$ è A, $R_{1,5}$ è B.

È possibile che il nodo A abbia già un certo numero di simboli in comune con il nodo entrante; in tal caso quest'ultimo prenderà da A le prime h + 1 righe della tabella, dove h è il numero di digits condivisi. Quando la procedura è terminata (i.e. la query ha dato esito positivo), verrà ricevuto dall'ultimo nodo, nel nostro esempio 36451467, il *Leaf Set* per inizializzare quello del nodo entrante.
Si può vedere come tale algoritmo di inizializzazione rispetti le proprietà di località, fintanto che le tabelle di *Routing* dei nodi, da cui

si sono ricevute le risposte, la rispettano; questo perché, assumendo che la metrica rispetti la disuguaglianza triangolare, i passi di *routing* minimizzano la distanza metrica percorsa localmente (nel passo corrente); quindi, producono una buona stima del percorso ottimo in termini di latenza: tali nodi raggiunti saranno buoni candidati per riempire la tabella del nuovo nodo (i nodi di B sono poco più lontani di quelli di A, suoi vicini, e così via).

Uscita di un nodo. Quando un nodo della rete fallisce o esce normalmente, è plausibile che sia presente nello stato interno di molti nodi, sia nella *Routing Table* sia nei *Set*. Occorre esaminare separatamente i tre casi e lo faremo nel caso in cui sia il nodo preso in esame finora a perdere la comunicazione con uno dei nodi a lui noti.

1. **Routing Table.** Quando un nodo viene identificato come non raggiungibile nella tabella di *routing*, questo procede con la Fase 3, come se la cella fosse vuota.
   Immediatamente, però, si cerca anche un rimpiazzo per tale cella contattando un nodo sulla stessa riga.
   Ad esempio, se fallisse 36440356 ($R_{3,4}$), verrebbe contatto il nodo 36430124 ($R_{3,3}$ o uno della medesima riga) chiedendo la sua entry $R_{3,4}$; se pure questo la avesse a *null*, la procedura avanzerebbe su $R_{4,i}$ fino ad ottenere un nodo valido.

2. **Leaf Set.** Nel caso in cui un nodo del *Leaf Set* non risponda ad un messaggio inviatogli, si procede a contattare l'estrema foglia della parte contenente il nodo inattivo. Ad esempio, se è 36451560 ad essere fallito, si contatterà 36451711, se fallisse 36451306 si contatterebbe 36451143.
   A questo punto, si ottiene da tale nodo il *set* di foglie e si sceglie, tra quelli non corrispondenti, un nodo adatto a sostituire l'inattivo.

3. **Neighborhood Set.** Viene aggiornato periodicamente con messaggi di *KeepAlive*, in quanto è raramente utilizzato durante il *routing*.
   Un nodo rilevato inattivo viene sostituito da un vicino di un proprio vicino, contattato appositamente.

È plausibile che un nodo malfunzionante, invece di non rispondere, notifichi la sua presenza per poi non propagare adeguatamente la ricerca. Per evitare configurazioni simili potrebbe essere necessario adottare meccanismi di *routing* più complessi, come, ad esempio, effettuare *multicast* sulla query.

2.7 Tecniche di Crittografia

La protezione dei dati in una social network è fondamentale: le architetture distribuite possono essere soggette a diversi tipi di attacchi: d'imitazione, di comunicazione e strutturali.

2.7.1 Attacchi d'Imitazione

Tra i possibili attacchi di questo tipo, i più noti sono i seguenti:

Furto d'identità: un utente non autorizzato prende possesso delle credenziali di un altro utente e agisce per suo conto.

Clonazione del profilo: utenti non autorizzati costruiscono una copia del profilo di un altro utente.

Porting del profilo: un utente non autorizzato crea un profilo utente su una *OSN* dove la vittima non è registrata.

Ad oggi non sono presenti specifiche contromisure contro questi tipi di attacchi; un aiuto alla prevenzione è portato dalla sensibilizzazione degli utenti nell'accettare amicizie solo da persone conosciute e nella gestione oculata delle informazioni personali.

2.7.2 Attacchi di Comunicazione e Strutturali

Per quanto riguarda questo genere di attacchi, i più frequenti sono:

Tracking delle comunicazioni: un utente non autorizzato intercetta una comunicazione fra due utenti.

Denial of Service (DoS): attacchi volti ad esaurire le risorse di un sistema in modo da impedire comunicazioni e altre azioni dell'utente.

Un metodo per limitare i problemi derivanti dagli attacchi descritti è quello di utilizzare la crittografia.

2.7.3 Crittografia Simmetrica

Processo di codifica che prevede **una sola chiave** condivisa sia nell'operazione di cifratura che di decifratura. Parametro fondamentale, per valutare la sicurezza della codifica, è la lunghezza della chiave. La crittografia simmetrica permette agli utilizzatori di servirsi di chiavi lunghe fino alla dimensione del messaggio.

La criticità del sistema risiede nello scambio della chiave, se non avviene in maniera sicura la robustezza è compromessa.

Un metodo per effettuare questa delicata operazione consiste nell'incapsulare la chiave in un'altra, generando la cosiddetta chiave Master da scambiare in un canale sicuro; in alternativa, si può inoltre utilizzare la crittografia asimmetrica.

2.7.4 Crittografia Asimmetrica

La crittografia asimmetrica, a differenza di quella simmetrica, prevede l'uso di **due chiavi**: una pubblica condivisa fra tutti coloro che interagiscono con l'entità proprietaria e una privata di cui solo il proprietario conosce il valore. Ogni utente è in possesso della coppia di chiavi, una privata e una pubblica, per questo il metodo è conosciuto anche come crittografia pubblica/privata. La criticità trovata nel processo di crittografia simmetrica, nel caso asimmetrico non è presente poiché non vi è bisogno di effettuare lo scambio di chiavi. Il sistema prevede che qualsiasi messaggio cifrato con la chiave pubblica di un'entità sia decifrabile solamente dalla chiave privata corrispondente.

Se un utente intende comunicare con un altro, cifra il proprio messaggio utilizzando la chiave pubblica del destinatario, che potrà così accedere al messaggio utilizzando la propria chiave privata.

Un altro esempio di utilizzo del processo di cifratura asimmetrica consiste nel fatto che il messaggio non sia ripudiabile: il proprietario del messaggio cifra quest'ultimo tramite la propria chiave privata, chiunque abbia accesso alla chiave pubblica del proprietario può decifrarne il contenuto. Il fatto che il messaggio possa essere decifrato tramite la chiave pubblica ne testimonia l'autenticità.

2.7.5 Attribute based Encryption

ABE [37] è un approccio recente che riconsidera il concetto di cifratura tramite chiave pubblica. Nei sistemi tradizionali un messaggio viene cifrato per uno specifico ricevente usando la sua chiave pubblica, con il metodo *Identity based Encryption* si permette alla chiave di essere una qualsiasi stringa, ad esempio l'indirizzo email del ricevente.

ABE definisce l'identità non in maniera atomica, ma come la composizione di un insieme di attributi, i messaggi potranno essere poi cifrati in base ad un sottoinsieme di questi attributi. Solo chi possiede una chiave che contiene gli stessi attributi può decifrare la comunicazione. Tale sistema è applicabile in un contesto come quello delle reti sociali utilizzando, come chiavi di cifratura, operazioni logiche su attributi riguardanti la relazione sociale tra mittente e destinatario, ad esempio "friend or friend of friend".

L'attribute based encryption comporta operazioni molto onerose dal punto di vista dell'efficienza.

2.7.6 Threshold Cryptography

La *Threshold Cryptography* [38] rappresenta un vero e proprio protocollo di cifratura, viene utilizzato nel caso in cui l'utente voglia condividere una risorsa con un gruppo di utenti fidati.

La chiave per effettuare la decodifica viene distribuita parzialmente a tutte le entità del gruppo, in modo tale che possa essere decifrato il contenuto se e solo se un sottoinsieme degli utenti è disponibile. La cardinalità del sottoinsieme viene definita soglia da cui deriva il nome del metodo.

Capitolo 3

DOP: Architettura del Sistema

In questo capitolo verrà discusso in che modo sono state utilizzate le tecnologie descritte nel secondo capitolo, per definire l'architettura del sistema proposto.
In primo luogo verranno descritti il linguaggio e i moduli dell'architettura *XACML* utilizzata per la definizione del nostro sistema. Successivamente verrà esposta l'architettura generale del sistema *Distributed Online social network Privacy* (*DOP*). Inoltre verrà presentato il meccanismo di accesso ai contenuti, la struttura del profilo utente e il processo di distribuzione dei contenuti alla disconnessione dell'utente.

3.1 XACML per il Controllo degli Accessi

L'**eXstensible Access Control Markup Language** è uno standard *OASIS*, che definisce un linguaggio per esprimere politiche di accesso ed è implementato in *XML*.

Le aziende utilizzano sistemi proprietari per la gestione delle autorizzazioni di accesso ad informazioni o applicazioni e spesso tali sistemi sono specifici per il dato o l'applicazione a cui fanno riferimento. Inoltre, è all'ordine del giorno utilizzare informazioni condivise in unità centrali, come cloud server, in cui è ancora più complessa la gestione degli accessi.

La necessità di uniformità nella gestione degli accessi ha portato alla creazione dello standard *XACML*[34]. La definizione delle *policy* sostituisce i sistemi per la gestione delle autorizzazioni: gli accessi non sono più basati solo sugli utenti, ma anche sulle risorse; una modifica della politica di gestione

si traduce in una modifica delle *policy* e non nella modifica dell'implementazione di un software.
Inoltre, le politiche sono implementate con un linguaggio semplice e diffuso universalmente come l'*XML*.

L'*XACML* è composto da diversi blocchi che contribuiscono a definire la seguente architettura:

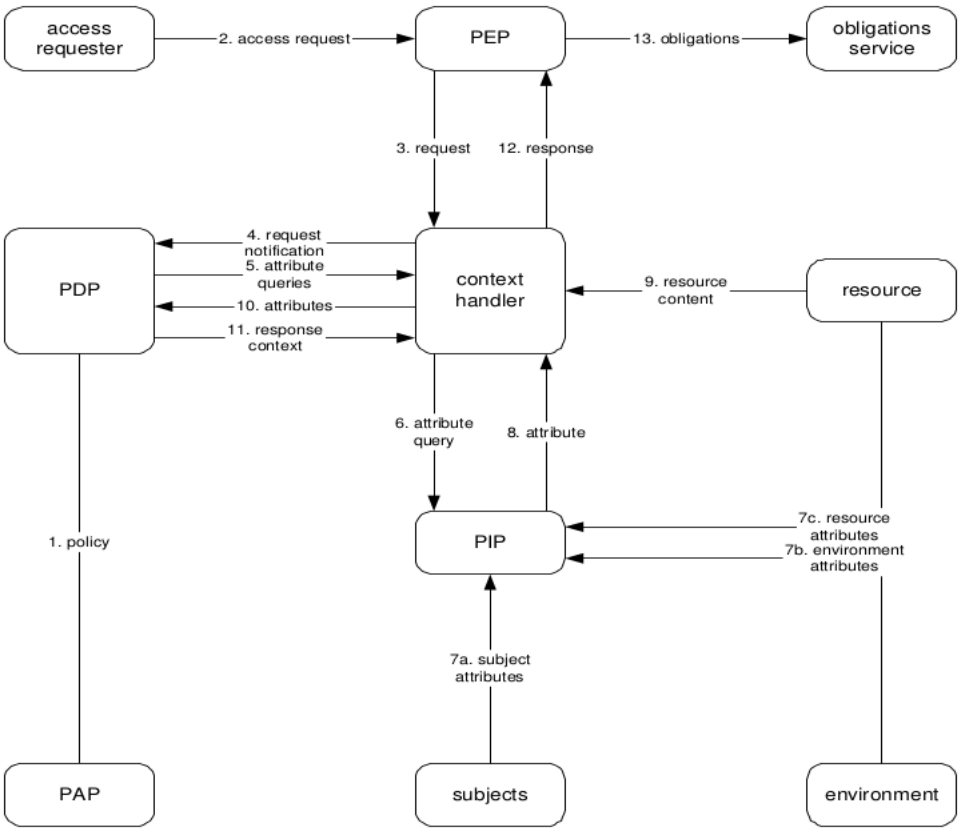


Figura 17. Struttura generale di un sistema di controllo accessi con *XACML*.

I moduli principali sono i seguenti:

Policy Enforcement Point (PEP): si occupa di ricevere le richieste ed inoltrare le risposte del sistema.

Context Handler (CH): converte le richieste ricevute in un dato linguaggio nel formato *XACML*, si coordina con i *PIP* per aggiungere infor-

mazioni alla richiesta e traduce le decisioni dal linguaggio *XACML* nel formato in cui era stata fatta la richiesta.

Policy Administration Point (PAP): entità in cui vengono salvate e gestite le policy.

Policy Information Point (PIP): se la richiesta ricevuta non è completa, il *CH* richiede al modulo *PIP* corretto di arricchire la richiesta con i dati mancanti.

Policy Decision Point (PDP): analizza le richieste e restituisce la decisione di autorizzazione.

L'accesso ad una risorsa avviene secondo la seguente procedura:

1. Le politiche vengono caricate nel *PAP* rendendole accessibili al *PDP*, in modo tale che all'arrivo di una richiesta possa valutarla sulla base della *policy* corretta.
2. L'arrivo di una richiesta viene presa in consegna dal *PEP*.
3. Il *PEP* inoltra la richiesta al *CH*, il quale dovrà convertirla in un linguaggio comprensibile al *PDP*.
4. Il *PDP* riceve la richiesta in *XACML* dal *CH*.
5. Se mancano degli attributi vengono interpellati uno o piu moduli *PIP*, passando dal *CH*.
6. Ogni *PIP* riceve la richiesta ed esegue un'analisi.
7. Il modulo *PIP* recupera gli attributi mancanti.
8. La richiesta arricchita viene rimandata al *CH*.
9. Il *CH* arricchisce ulteriormente la richiesta, se possibile.
10. Il *CH* recapita la richiesta modificata al *PDP* che, a questo punto, può valutarla.
11. Il *PDP* seleziona, fra le *policy* disponibili nel *PAP*, quella da applicare e restituisce al *CH* l'*XACML response context*.

12. Il *CH* traduce la decisione e la inoltra al *PEP*.

13. Il PEP infine da il permesso o lo nega al richiedente in base al contenuto dell'*XACML response context* e fa applicare le obbligazioni se ve ne sono.

XACML fornisce un metodo per definire delle obbligazioni allegate alle *policy*, che devono essere compiute una volta che è stata valutata una richiesta.
Nel caso in cui fossero presenti delle obbligazioni, il *PEP*, una volta ricevuto il risultato della valutazione della richiesta, ha il compito di assicurarsi che vengano eseguite.

3.1.1 Regole

Le regole sono le entità più elementari di una *policy*.

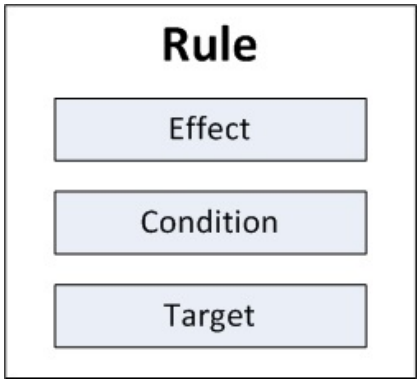


Figura 18. Struttura di una regola *XACML*.

Una regola viene valutata sulla base dei suoi componenti, che sono:

target,

effetto,

condizione,

obbligazioni,

avvertenze.

```
Policy -
<Rule Effect="Permit" RuleId="Rule_On_Withdrew">
 <Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Money</AttributeValue>
      <ResourceAttributeDesignator AttributeId="http://domain.wso2.org/resources/parents"
                                   DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </Apply>

    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Withdrew</AttributeValue>
      <ActionAttributeDesignator AttributeId="http://domain.wso2.org/actions/parents"
                                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </Apply>
  </Apply>
 </Condition>
</Rule>
```

Figura 19. Semplice regola con solo la condizione.

In Figura 19 viene evidenziato come sia possibile definire una regola solamente con una delle sue componenti; in questo caso si tratta della *Condition*, la quale ha lo scopo di consentire l'accesso solo se vengono individuate nella richiesta la coppia di parole chiave "Money" e "Withdrew".

Figura 20. Struttura di un target *XACML*.

Il **target** definisce l'insieme di richieste alle quali la regola deve essere applicata, che può essere poi raffinato dalla **condizione** booleana. Il *PDP* si occupa di verificare che gli attributi della richiesta corrispondano al *target* specificato dalla regola. Il *target* può essere omesso da una regola, in tal caso è considerato il *target* della *policy*.

L'**effetto** di una regola può essere *Permit* o *Deny*, cioè quale si vuole che

sia la conseguenza della valutazione.

Le **obbligazioni** sono operazioni che devono essere svolte subito dopo la valutazione.

Le **avvertenze** sono facoltative e possono essere aggiunte ad una regola, nel caso in cui si intenda comunicare al *PEP* informazioni aggiuntive, che possono essere ignorate da esso.

3.1.2 Policy

Una **policy** incapsula più regole in modo tale da poterle scambiare tra le entità del sistema.

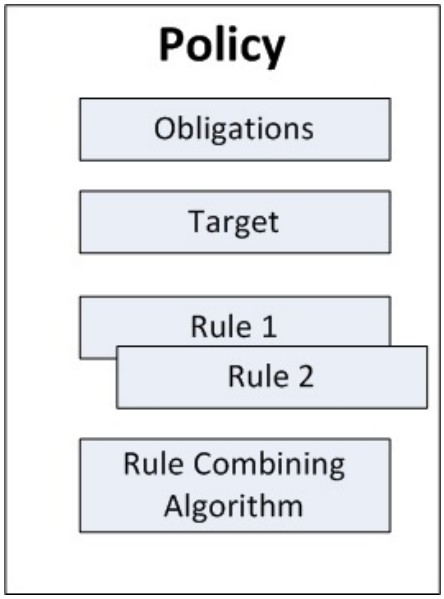


Figura 21. Struttura di una politica *XACML*.

Esse sono mantenute e organizzate dalla componente *PAP* e sono composte dai seguenti elementi:

- target,
- combining-algorithm per regole,
- un insieme di regole,

- obbligazioni,
- avvertenze.

Il **target** di una *policy* può essere calcolato in due modi differenti. Il primo consiste nel generare l'unione di tutti i target delle componenti inferiori; alternativamente, può essere determinato dall'intersezione di tali elementi.

Attributi

Gli attributi relativi al soggetto o alla risorsa nelle richieste d'accesso possono essere identificati in due modi:

AttributeDesignator contenente una *URN*[1] che specifica univocamente l'attributo;

AttributeSelector contenente un'espressione *XPath*[2] che definisce l'attributo tramite la sua locazione all'interno del *context*.

XACML supporta attributi multivalore, più precisamente delle collezioni che prendono il nome di **bag**. Una *bag*, a differenza di un insieme, può contenere elementi duplicati. Il sistema fornisce un insieme di funzioni in grado di gestire questo tipo di attributi.

Le politiche possono essere definite e valutate da persone diverse e in diverse parti: *XACML* permette alle *policy* di fare riferimento a politiche esterne o di contenerne al proprio interno.

Operatori

Al fine di valutare le richieste di accesso, può essere necessario utilizzare degli operatori matematici, ad esempio per confrontare due valori numerici o operatori sugli insiemi.

XACML mette a disposizione un insieme di funzioni pronte e la possibilità di definire funzioni *custom*, tramite l'uso dell'elemento **Apply**. L'elemento *Apply* possiede un attributo *XML*, chiamato *FunctionId*, il quale specifica le funzioni da applicare ai contenuti dell'elemento.
Sono presenti poi operatori per le date, il tempo e le durate.

[1] Uniform Resource Name, un *URI* (Identifier) che non permette di localizzare la risorsa.
[2] *XPath* è un linguaggio parte della famiglia *XML* che permette di individuare i nodi all'interno di un documento *XML*

3.1.3 Policy Combination

La *policy* applicabile potrebbe essere la composizione di più regole o di più politiche.
Gli elementi che costituiscono una *policy* sono l'insieme delle regole e una procedura per combinarle. Un insieme di *policy* e la procedura che specifica come comporle costituiscono il **PolicySet**. L'utilizzo dei *PolicySet* permette al *PDP* di valutare l'applicabilità di politiche composte.

Combining Algorithms

XACML mette a disposizione un insieme di algoritmi, espressi come regole, volti a definire in che modo attuare la valutazione se sono presenti più regole o più *policy*.

Gli esempi standard comunemente utilizzati per gestire le politiche composte sono:

1. *Deny-overrides*
2. *Permit-overrides*
3. *First-applicable*
4. *Only-one-applicable*

Nel primo caso, se una delle regole/policy non è applicabile, l'intera valutazione risulterà essere negativa, al contrario nel secondo algoritmo basta trovare una regola/policy applicabile per restituire esito positivo.

First-applicable è la regola per la quale si valuta la prima regola/politica applicabile della lista.

Only-one-applicable si applica solo alle *policy*, non alle regole, e garantisce che nella lista delle politiche che compongono il *PolicySet* sia presente una sola politica applicabile. Se più di una politica è applicabile il risultato restituito dal *PDP* è indeterminato; se nessuna delle *policy* o *PolicySet* si applica allora risulta non applicabile. Solo quando esattamente una delle *policy* può essere applicata, il *PDP* restituisce la valutazione di quest'ultima.

Le decisioni di autorizzazione possono avvenire in base a qualche caratteristica del soggetto che tenta l'accesso, tra cui la sua identità, oppure in base alle caratteristiche della risorsa a cui accedere.

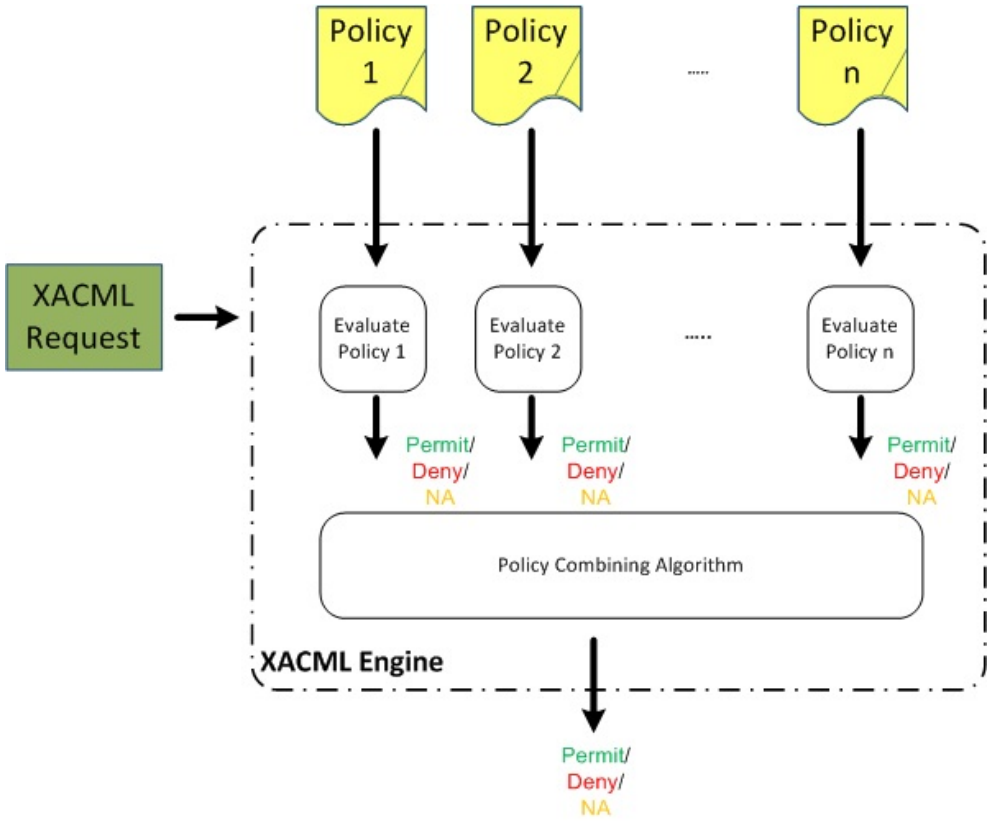


Figura 22. Schema di valutazione di una richiesta *XACML*.

Come si può vedere dall'immagine, il *combining algorithm* viene eseguito quando la valutazione di tutte le *policy* inerenti la richiesta ha dato esito.

3.1.4 Livello di Astrazione

La struttura *XACML* non definisce nello specifico il *Policy Enforcement Point*, in diversi sistemi non è detto che questa funzione venga compiuta dall'entità che gestisce gli altri moduli. Un *PEP* potrebbe essere, ad esempio, un *gateway* remoto o parte di un *Web-server*.

XACML astrae dalle modalità con cui la richiesta viene espressa; ovviamente vi è la necessità di un elemento intermedio che svolga la funzione di convertitore della richiesta e della risposta, dal linguaggio nativo in *XACML* e viceversa. In questo modo le politiche possono essere scritte e valutate

indipendentemente dallo specifico ambiente in cui vengono applicate.

Obbligazioni

In molte applicazioni le politiche specificano azioni che devono essere svolte in concomitanza con la valutazione. Tali azioni sono contenute nell'elemento **Obligations**; dato che non è previsto uno standard per la definizione delle obbligazioni, se l'*Enforcement Point* non riesce ad interpretare ed applicare tutte le azioni in esso definite il risultato della valutazione avrà esito negativo.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
  xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
  http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
  PolicyId="urn:oasis:names:tc:xacml:3.0:example:SimplePolicy1"
  Version="1.0"
  RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
  <Description>
    Medi Corp access control policy
  </Description>
  <Target/>
  <Rule
    RuleId= "urn:oasis:names:tc:xacml:3.0:example:SimpleRule1"
    Effect="Permit">
    <Description>
      Any subject with an e-mail name in the med.example.com domain
      can perform any action on any resource.
    </Description>
    <Target>
      <AnyOf>
        <AllOf>
          <Match
            MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string"
              >med.example.com</AttributeValue>
          <AttributeDesignator
            MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject"
            AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
  </Rule>
</Policy>
```

Figura 23. Esempio di *policy* in *XACML*.

Come si può vedere dall'esempio in Figura 23, ogni documento di una *policy* deve contenere una riga che descrive la codifica *XML* e tutto il contenuto è racchiuso all'interno di un *tag Policy* che ha come attributi la definizione della specifica dello schema, un identificatore per la politica e l'algoritmo di *rule-combining* che deve essere applicato (in questo caso è utilizzato il *deny-overrides* già descritto in precedenza).

La *Description* opzionale permette di definire un nome per la *policy* a scopo di leggibilità. In seguito, è espresso il target della *policy*, che nel nostro esempio è lasciato vuoto, in quanto specificato in ogni regola del file.

Le regole sono elencate subito dopo, con un identificatore, l'effetto e la descrizione facoltativa; viene, quindi, definito il target della regola che può essere intersezione (*AllOf*) o unione (*AnyOf*) di un insieme di *Match*.

Nel nostro esempio abbiamo solo una condizione di *Match*, che comprende un identificatore, *AttributeValue* e *AttributeDesignator*. L'*AttributeValue* è composto, oltre che dal valore effettivo dell'attributo, anche dal tipo[3]. L'*AttributeDesignator* stabilisce quale è l'attributo che deve essere confrontato, in questo caso è il soggetto.

[3]Il tipo degli attributi è rappresentato da un *URI*, che deve essere valido per lo schema *XACML*.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
  ReturnPolicyIdList="false">
  <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject">
    <Attribute IncludeInResult="false"
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
      <AttributeValue
        DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
         >bs@simpsons.com</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes
    Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
    <Attribute IncludeInResult="false"
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
         >file://example/med/record/patient/BartSimpson</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes
    Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <Attribute IncludeInResult="false"
        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
         >read</AttributeValue>
    </Attribute>
  </Attributes>
</Request>
```

Figura 24. Esempio di richiesta *XACML*.

La struttura generale di una richiesta *XACML* si presenta simile a quella di una *policy*; a differenza di quest'ultima, non contiene target e regole, quindi non è necessario definire algoritmi per la combinazione delle regole. Consiste, come si può vedere in Figura 24, di una lista di attributi con associato un valore, che saranno poi confrontati con quelli contenuti nella *policy*. Ogni attributo comprende identificativo, tipo e valore.

3.2 DOP: Struttura del Sistema

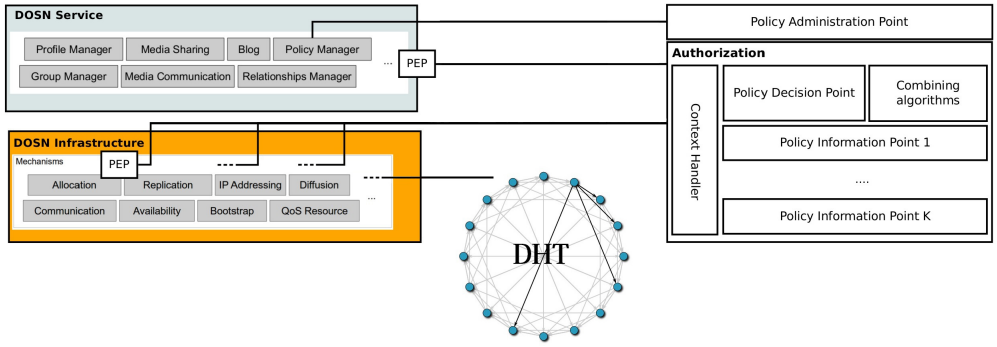


Figura 25. Architettura del sistema.

Il sistema **DOP** ha lo scopo principale di offrire un supporto per garantire la privacy degli utenti di una *DOSN*, permettendo loro di definire politiche di controllo degli accessi ai loro contenuti tramite linguaggi avanzati (i.e., *XACML*), ed allocando i profili degli utenti che si disconnettono dalla *DOSN* su nodi di utenti che sono autorizzati ad accedere a tali profili secondo la politica precedentemente definita.

L'idea alla base del nostro approccio è che l'utente che ospita sul suo nodo i dati che rappresentano il profilo di un altro utente non possa ricavare informazioni aggiuntive accedendo direttamente a tali dati perché otterrebbe le stesse informazioni alle quali ha già accesso tramite l'interfaccia della *DOSN*.

In Figura 25 è rappresentata l'architettura del sistema *DOP* che, per preservare la privacy degli utenti come precedentemente descritto, prevede l'integrazione di un sistema avanzato di autorizzazione, basato su *XACML*, nell'architettura del nodo della *DOSN*. In particolare, sulla sinistra è presente l'architettura di un nodo della rete sociale distribuita composta da **due livelli**: il **livello infrastrutturale** in cui sono presenti servizi per l'avvio del sistema, l'allocazione e la replicazione dei contenuti e il **livello di servizio**, con tutte le funzionalità offerte agli utenti, tra cui la gestione del profilo e delle amicizie.

L'architettura fa uso della *Distributed Hash Table* ***Pastry***, per organizzare al meglio il recupero delle informazioni relative ai nodi su cui vengono memorizzate le repliche dei profili quando un nodo si disconnette dalla rete.

Nella parte destra della Figura 25 è presente il sistema di autorizzazione

basato sullo standard di riferimento *XACML* mostrato nel paragrafo precedente, che interagisce con entrambe i livelli della *DOSN* (servizio ed infrastruttura). Quindi, il sistema di autorizzazione viene invocato sia quando un utente richiede di accedere ai contenuti di un altro utente, che quando deve essere individuato il nodo più opportuno per allocare il profilo di un utente che si vuole disconnettere.

Come si può notare in Figura 25, il *PEP* (*Policy Enforcement Point*) è il modulo *XACML* che agisce da interfaccia con la *DOSN* e viene utilizzato da entrambi i livelli. A livello di servizio, il *PEP* intercetta le richieste degli utenti esterni di accedere ai profili allocati sul nodo e le passa al sistema *XACML*, che verificherà l'ammissibilità di tale richiesta.

A livello infrastrutturale, al fine di garantire la continua presenza sulla *DOSN* del profilo di un utente che si disconnette dalla rete, si deve operare in modo tale che le informazioni sul profilo dell'utente uscente non vadano perse. A questo scopo, il livello infrastrutturale simula un accesso ai contenuti, tramite il *PEP*, da parte dei contatti online per stabilire chi rispetta le politiche di privacy dell'utente. Tra i contatti che soddisfano la politica, il sistema sceglie un contatto anche in base ad informazioni provenienti dal livello infrastrutturale, come durata media delle sessioni o caratteristiche fisiche (banda, accessibilità) del nodo su cui è ospitato l'utente.

È necessario prevedere la possibilità di trovarsi in uno scenario in cui nessun amico dell'utente uscente sia connesso al sistema: in tale eventualità si prevede che le informazioni vengano memorizzate in modo cifrato su nodi *untrusted*, ad esempio sui nodi della *DHT*. Tuttavia, questo caso non verrà trattato nella tesi.

Nella prima versione di *DOP*, il **profilo utente** è stato considerato come una singola entità si fini della riallocazione. Successivamente, si è pensato di strutturare il profilo come una **struttura gerarchica ad albero**, per dare la possibilità all'utente di definire singole politiche di accesso per ogni elemento dell'albero creando così una gerarchia di politiche di *privacy*. La strutturazione del profilo ad albero ha portato ad una diversa distribuzione dei contenuti dell'utente al momento della disconnessione. Risulta, infatti, possibile affidare parti del profilo dell'utente che si disconnette ad utenti diversi, mantenendo sempre disponibili tutte le informazioni contenute nel profilo.

3.2.1 Accesso al Profilo

L'accesso ai contenuti di un utente può avvenire a seguito di una normale interazione oppure successivamente ad una disconnessione.

Il primo caso è ciò che avviene nelle OSNs siano esse distribuite o meno; l'utente A pubblica dei contenuti, un altro utente B, per un determinato scopo, intende vedere i contenuti di A ed invia la richiesta di accesso.
La richiesta viene valutata secondo le politiche di privacy definite dal possessore del contenuto, se B ha i diritti necessari ad accedere all'informazione ad esso viene garantito l'accesso. Se l'esito della validazione risulta negativo, invece, l'utente non può accedere al contenuto.

Nel capitolo successivo mostreremo alcune politiche caratterizzate dal riferire attributi diversi dell'utente. A partire dalla prima politica, che riguarda solo la presenza di un'amicizia tra il possessore del contenuto e chi vi accede, si definiscono quindi politiche che prendono in considerazione il numero di amici comuni, la forza del legame di amicizia e la locazione geografica degli utenti.

Nel secondo caso, l'accesso viene simulato al fine di individuare il nodo o i nodi su cui devono essere manenuti i contenuti dell'utente uscente nel sistema. Supponiamo che l'utente A si disconnetta. Fra la lista degli utenti amici di A si scelgono i contatti online in quel determinato momento e si simula un accesso ai contenuti del suo profilo. Le informazioni vengono quindi affidate al contatto più adatto, tale contatto (B) viene scelto in base a diverse caratteristiche:

- deve soddisfare le politiche di privacy specificate da A
- si considera la durata media delle sessioni di B o qualche altra misura statistica relativa alla attività online di esso.
- altre caratteristiche, come la banda a disposizione di B, la distanza geografica tra A e B o l'accessibilità di B (presenza o meno di NAT).

Una volta effettuata la distribuzione del profilo, il sistema deve inoltrare le richieste d'accesso, dirette ad A, all'utente che possiede le informazioni.

Solo nel caso improbabile, ma non impossibile, in cui nessun utente amico sia online, il profilo viene cifrato e memorizzato sulle DHT.

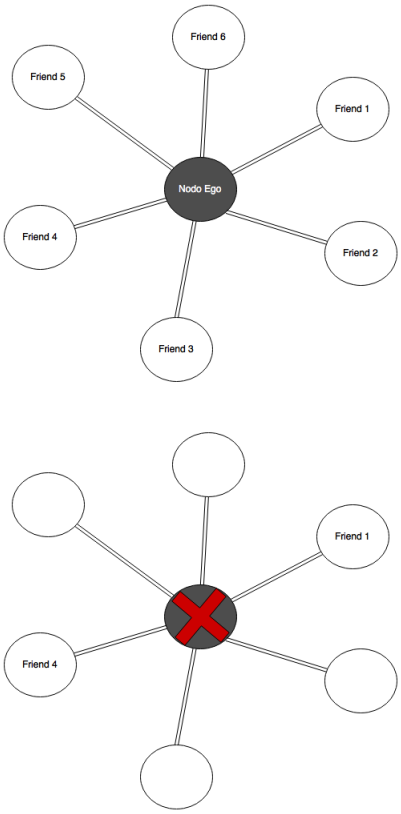


Figura 26. Ego Network con successiva disconnessione dell'Ego.

Nella Figura 26 è possibile vedere come, nel caso in cui un nodo si disconnetta dalla rete, vengano selezionate alcune repliche tra i suoi vicini (in questo caso Friend 1 e Friend 4).

La *DHT*, nel nostro caso *Pastry*, è implementata da tutti i nodi che partecipano alla social network e memorizza informazioni relative ai nodi che siano, in un certo istante, offline ed eventualmente l'identificatore del nodo offline N è la chiave di accesso alla *DHT* e permette di identificare il nodo della *DHT* che gestisce le informazioni relative ad N nel periodo in cui questo si trova offline.

Nella *DHT* verrà aggiornato il nodo che si occupa di gestire i dati dell'Ego che si è appena disconnesso, inserendo i riferimenti ai nodi che hanno ottenuto le repliche dei suoi dati, come illustrato in Figura 27. È importante notare che le informazioni del nodo che si è disconnesso sono memorizzate su un nodo della *DHT*, che non necessariamente rappresenta un suo amico e quindi, in

generale, è un nodo *untrusted*. Per questa ragione, tali informazioni devono essere memorizzate in modo cifrato.

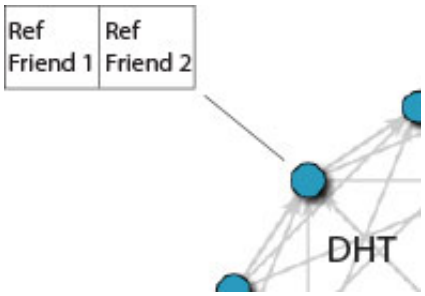


Figura 27. Aggiornamento dei riferimenti del nodo sulla *DHT*.

3.2.2 Struttura del Profilo

Il profilo di un utente può essere visto come un singolo dato complesso, oppure può essere inteso come un insieme di informazioni che possono essere descritte in maniera gerarchica tramite l'uso di un albero.

L'utente di una *OSN* mette a disposizione della rete diverse informazioni, in primo luogo informazioni personali come nome, cognome, linguaggio, email, genere, stato sentimentale, luogo e data di nascita. In seguito inserirà informazioni legate agli interessi, allo sport, alla locazione corrente e a quella relativa alla sua residenza.

Parte del profilo è la lista delle amicizie dell'utente con cui interagisce e tutti i messaggi delle comunicazione private intraprese, sia in entrata che in uscita.
Un ulteriore metodo di interazione sono i post pubblicati con i *like* e i commenti associati ed a differenza dei messaggi personali i post vengono condivisi con le persone che hanno i diritti di visibilità per accedervi. Inoltre il profilo comprende anche gli album e le immagini sulle quali è possibile esprimere commenti oppure *tag*[4]. L'utente che inserisce un'immagine nel suo profilo può inserire un *tag* associato ad un amico, ciò significa che il contenuto sarà visibile anche agli amici dell'amico. Possono nascere problemi se chi ha pubblicato l'immagine non vuole che essa sia visibile ai friends of friends.

[4]un attributo di un media che permette di individuare le persone in esso presenti

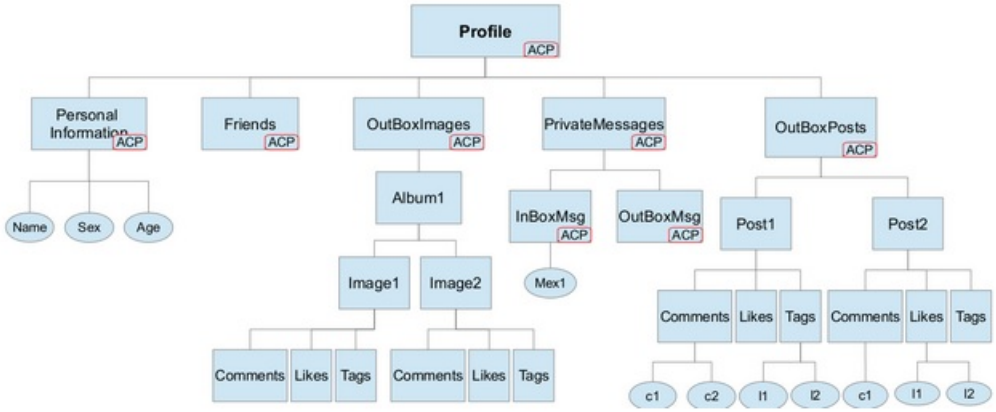


Figura 28. Strutturazione gerarchica del profilo di un utente della DOSN.

Nella seconda versione di *DOP* abbiamo quindi definito il profilo dell'utente come una struttura gerarchica, in cui la radice rappresentata il riferimento logico ai contenuti del profilo, al secondo livello dell'albero troviamo:

- i dati personali,
- la lista degli amici,
- le conversazioni private,
- i post ricevuti/postati,
- gli album delle immagini.

I livelli sottostanti, sono relativi alla lista degli album, che a loro volta conterranno le singole immagini, il registro delle conversazioni private (input e output) e l'insieme dei post.

Le foglie sono rappresentate dai commenti, *like* e *tag* presenti nelle immagini e nei post. Ogni elemento dell'albero viene identificato univocamente da un **ID**, che risulta essere l'identificativo del padre concatenato ad una stringa che differenzia il nodo dai fratelli.

I fattori che rendono diverso *DOP* dai sistemi descritti nel Capitolo 2 sono dati principalmente dalla maggiore espressività nella definizione delle *policy*, grazie al linguaggio *XACML*, dalla maggiore *granularità* con cui è

possibile associare le politiche agli elementi e dalla strutturazione gerarchica del profilo. È possibile infatti specificare politiche sul singolo nodo dell'albero oppure sull'intero sotto-albero.

Supponiamo che un utente A definisca una politica generale per gli accessi al profilo, che costituisce la radice dell'albero, che permetta la visualizzazione dei contenuti a tutti i suoi amici. I contatti in relazione d'amicizia con A possono quindi accedere a tutti i dati di quest'ultimo. Successivamente l'utente A specifica una diversa politica per accedere alle sue informazioni personali, che potranno ora essere visualizzate solo dagli amici che abitano nella sua città. Gli amici di A che non risiedono nella sua città non hanno più i diritti necessari per visualizzare le informazioni personali di A. Allo stesso modo A può decidere che uno dei suoi album o una delle sue immagini possa essere visibile solamente alla sua famiglia.

In questo modo, si crea una gerarchia fra le politiche definite dall'utente: più ci si avvicina alle foglie più la politica è **vincolante**. La politica più generale rimane sempre quella relativa al profilo, le *policy* specificate per i livelli sottostanti devono essere più restrittive di quella relativa al profilo. In generale la politica definita per un certo livello dell'albero deve essere più generale di tutte quelle specificate per gli elementi del sotto-albero.

3.2.3 Distribuzione Contenuti

La suddivisione del profilo nella struttura ad albero ha reso più complessa la fase di distribuzione dei contenuti dell'utente uscente.

Considerando il profilo come un'entità unica, vi è bisogno di un solo contatto che si occupi di mantenerlo interamente. Strutturando ad albero le informazioni, è possibile partizionare il profilo affidando i sotto-alberi a contatti diversi.

Prendendo in esame l'esempio precedente, supponiamo che l'utente A specifichi politiche per le sue informazioni personali e per una delle sue immagini.
Ovviamente non è possibile affidare l'intero profilo di A ad una persona che non faccia parte della sua famiglia e che non viva nella sua città, poiché non sarebbe accettabile per le politiche di privacy definite. Risulta invece possibile dividere l'albero, affidando, ad esempio, la parte delle informazioni personali ad un utente, l'immagine per cui è stata specificata una *policy* ad un altro e il resto ad un altro utente ancora.

Il partizionamento dell'albero risulta necessario innanzitutto perchè po-

trebbe non essere possibile affidare l'intero profilo ad un unico utente, rispettandone le politiche di gestione degli accessi ai contenuti. In secondo luogo, potrebbe essere importante considerare il **comportamento temporale** degli utenti: invece di affidare i dati ad una sola persona che ha i diritti per visualizzarli, ma che risulta non essere molto attiva nel sistema (poche sessioni online e di breve durata), si preferisce assegnare ogni parte del profilo all'utente con il comportamento temporale migliore. Il nuovo sistema di identificazione non solo è necessario per la ricerca dei contenuti all'interno del profilo, ma inoltre rende possibile le richieste di accesso alla singola informazione e, di conseguenza, la stesura di *policy* specifiche per un singolo dato. Il profilo strutturato, come descritto, permette la migrazione partizionata dei contenuti: sotto-alberi diversi possono venire assegnati ad utenti diversi.

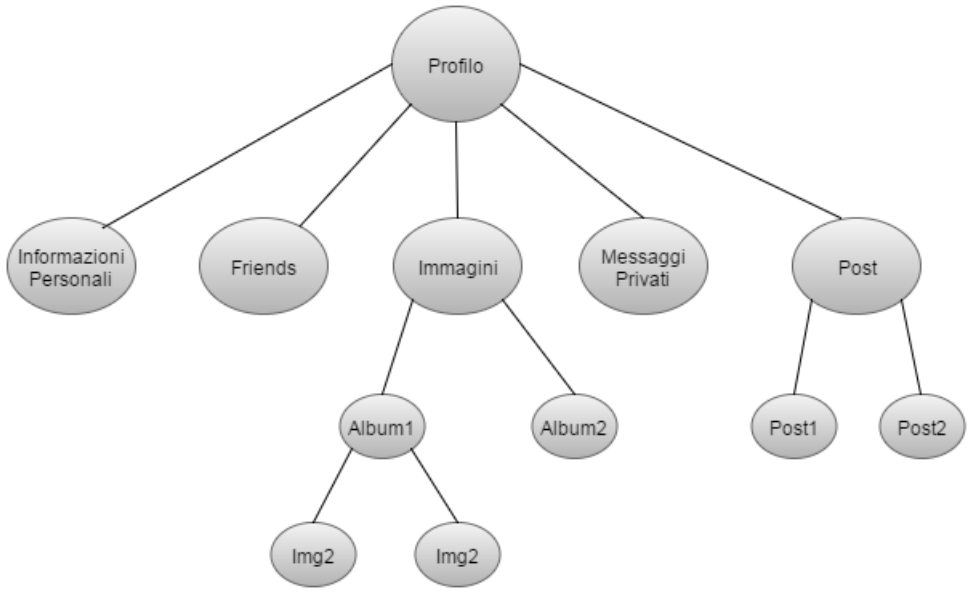


Figura 29. Esempio di un profilo gerarchico.

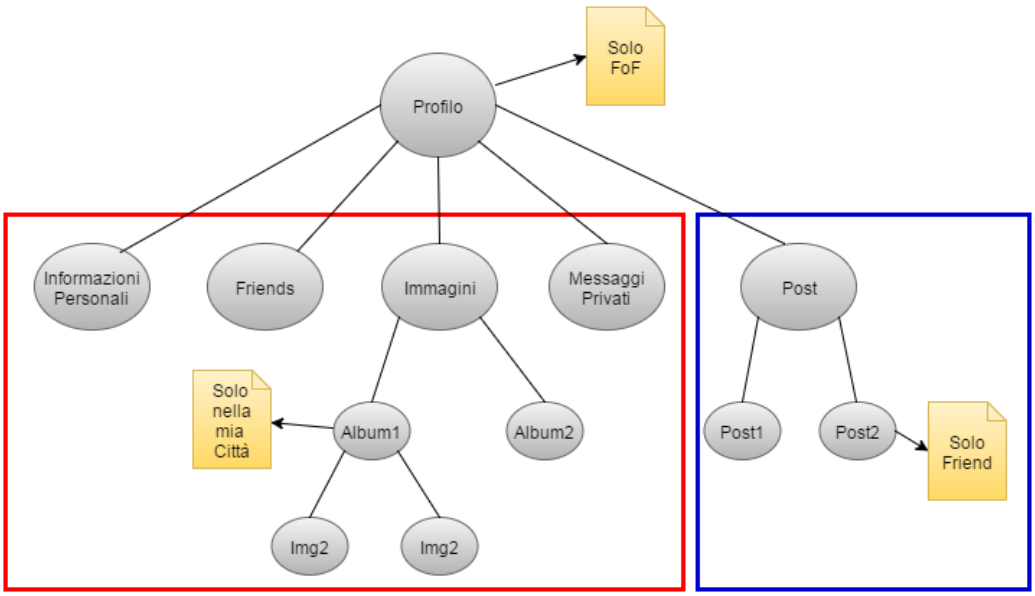


Figura 30. Possibile partizionamento del profilo gerarchico.

In Figura 30 è mostrato lo stesso albero di Figura 29 a cui è stato applicato un partizionamento. Nell'albero sono presenti tre policy: una generale che permette la visualizzazione del profilo da parte degli amici di amici (*Friends of Friends*), una associata all'Album1, che rende possibile l'accesso al sotto-albero agli amici della stessa città dell'utente, e l'ultima relativa al Post2, che permette l'accesso degli amici dell'utente. È possibile che il profilo venga suddiviso in partizioni se alcuni degli amici online dell'utente non ne hanno accesso completo. Nel caso in figura, l'utente che memorizzerà i contenuti all'interno del rettangolo rosso ha sicuramente accesso all'intero profilo, poiché soddisfa la *policy* più restrittiva. Le informazioni restanti, racchiuse nel rettangolo blu, possono essere migrate ad un secondo utente se quest'ultimo risulta avere prestazioni (sessioni online, banda etc.) migliori.

3.2.4 Integrazione di XACML in DOP

Per realizzare il modulo nella parte destra di Figura 25, abbiamo individuato un insieme di strumenti adatti all'implementazione dei moduli *XACML* Il controllo degli accessi viene gestito implementando in linguaggio *Java* i moduli che fanno parte dell'architettura di *XACML*.

Non vi è stata la necessità di implementare il *Context Handler* (*CH*), dato che le richieste vengono formulate e recapitate direttamente nel formato *XACML*.

Il modulo decisionale, il *Policy Decision Point* (*PDP*), è implementato dal software open source *Java* **Balana** [5]. Tale software si basa sull'implementazione *Sun's XACML*, supporta *XACML* 3.0, permette l'istanziazione di un modulo *PDP*, dando la possibilità di configurazione all'utente. I parametri di default sono modificabili tramite la stesura di un file di configurazione in cui è possibile specificare, oltre ad alcuni paramentri per l'esecuzione, gli **AttributeFinderModule** e altri moduli, in base alle esigenze.

Ogni tipo di *policy* definita richiede il suo specifico modulo *PIP*, con il compito di arricchire la richiesta recuperando le informazioni relative ai profili degli utenti in causa. Nella maggior parte dei casi il processo di arricchimento è necessario. I moduli *Policy Information Point*, all'interno della simulazione, vengono implementati come estensioni degli *AttributeFinderModule*.

Il *PAP* (*Policy Administration Point*) è un modulo che si occupa dell'organizzazione, gestione e recupero delle *policy* specificate dall'utente.

[5] wso2.com/products/identity-server/

Le politiche mantenute devono essere salvate all'interno del *file-system* dell'utente che le ha definite, per poi essere eventualmente trasferite nella macchina di un altro nodo nella fase di riassegnamento delle risorse (insieme al profilo). Il modulo *PAP* ha il compito di recuperare le politiche dal disco e, a tale scopo, viene definito un protocollo per l'estrazione dei file dalla memoria, specificato nella libreria *Balana*; tale strumento può essere esteso e modificato per ottenere diversi protocolli di archiviazione e recupero delle *policy*, ma non è risultato necessario nell'interesse del progetto di tesi.

Il sistema predefinito per il recupero delle politiche dalla memoria si appoggia all'implementazione del *file-system* per il sistema operativo in uso. Questo modulo viene inizializzato assieme ai protocolli di ogni nodo della rete.

Capitolo 4

DOP: Implementazione del Sistema

4.1 Introduzione

La tesi ha previsto l'implementazione di DOP mediante una simulazione che utilizza diversi strumenti: **Balana**[1], **PeerSim**[2] e **Pastry** [36].

PeerSim è la libreria che ha permesso l'implementazione della simulazione mediante un motore a cicli e la definizione dei partecipanti alla simulazione.

Balana ha la funzione di *Policy Decision Point*, verifica se le richieste di accesso alle risorse siano conformi alle politiche definite dall'utente proprietario dei contenuti.

Pastry è la *DHT* scelta nella simulazione. L'implementazione di *Pastry* utilizzata è quella sviluppata appositamente per *PeerSim* e prende il nome di *MSPastry* [35].

Gli strumenti elencati sono implementati o compatibili con il linguaggio *Java*, linguaggio utilizzato per l'implementazione della simulazione. *PeerSim* e *Balana* sono entrambi framework open-source sviluppati in *Java*.

Inoltre, la simulazione utilizza un insieme di dati raccolti tramite il software *Social Circles* organizzati in file in formato *CSV*.

Nello sviluppo del sistema si è prevista una prima fase in cui sono state implementate politiche di accesso per il profilo, inteso come un'entità unica. Dopo aver eseguito alcuni test su questa prima versione, si è passati ad una

[1] wso2.com/products/identity-server/

[2] http://peersim.sourceforge.net/

estensione che definisce politiche a grana più fine, in particolare queste ultime sono state specificate per ogni singolo contenuto all'interno della struttura. Questa evoluzione non solo è stata definita allo scopo di estendere il progetto, ma anche per rendere più realistica la strutturazione e per poter condurre test di confronto tra le due versioni.

4.2 Descrizione e Uso del Simulatore

Il software implementato è una simulazione effettuata tramite l'uso della libreria *Java* **PeerSim**.

PeerSim è uno strumento che permette la simulazione di una rete *peer-to-peer*, tenendo in considerazione la larga scalabilità e la dinamicità di tali reti. Rilasciato al pubblico tramite le licenze open source *GPL*, è composto da due motori di simulazione: **a cicli** o **a eventi**. Nel primo caso la simulazione consiste nell'eseguire in sequenza i protocolli associati ai nodi della rete, mentre nel secondo l'esecuzione dei protocolli è guidata dal verificarsi degli eventi.

PeerSim viene eseguito su un singolo host in maniera *single-threaded*, non sfruttando le architetture multi-core, ed è caratterizzato da un nucleo minimo di classi che realizzano le funzionalità di base della simulazione. Affiancate a questo nucleo sono presenti le componenti rappresentate dalle classi definite dall'utente, i cui riferimenti vengono esplicitati nel file di configurazione. Le componenti specificabili dall'utente sono i controlli, i protocolli e i nodi.

L'interfaccia *Control* permette di implementare classi per l'esecuzione delle operazioni che richiedono la conoscenza globale della rete, esistono tre diversi tipi di controlli: di inizializzazione, osservatori oppure dinamici. I controlli possono essere eseguiti ad ogni iterazione, nel caso siano dinamici o osservatori, oppure una sola volta inizialmente, nel caso siano d'inizializzazione.

I nodi rappresentano le entità che partecipano alla rete *peer-to-peer* e, in questo caso, corrispondono al profilo degli utenti che utilizzano il servizio *DOSN*.

I protocolli rappresentano il comportamento dei nodi ad ogni iterazione della simulazione, le operazioni svolte da ogni nodo. L'utente può definire più di un protocollo, anche se ai fini della simulazione ne è necessario uno solo.

4.3 L'Ambiente di Simulazione

La prima operazione compiuta da *PeerSim* è quella di effettuare il *parsing* di un file di testo, nel quale è possibile definire in che modo si vuole configurare il simulatore. In particolare è possibile definire i parametri generali come il numero di cicli, il seed e le dimensioni della rete. La sintassi per l'impostazione del file di configurazione è strutturata per righe nel formato seguente:

componente.attributo valore

Le componenti usate sono: *network*, *simulation*, *control*, *init*, *order* e *protocol*; il valore viene assegnato all'attributo associato.

Esempio di configurazione dei parametri generali:

- network.size 500
- simulation.cycles 30
- random.seed 123456

Ogni motore ha degli attributi specifici: *cycles* fa riferimento solo al motore *cycle-driven*. Sempre nel file di configurazione si rendono note le classi che verranno considerate controlli e quali di questi sono dinamici, osservatori o di inizializzazione.

I controlli dinamici sono quelli che intervengono sulla rete apportando delle modifiche alla rete stessa, gli osservatori sono anch'essi eseguiti ad ogni ciclo, ma si limitano ad effettuare calcoli sulle informazioni ottenute dalla rete, senza effetti laterali. Il nucleo della simulazione consiste nella definizione delle classi che implementano il comportamento dei nodi della rete mediante i protocolli. In maniera opzionale vi è poi la possibilità di specificare in quale ordine debbano essere eseguiti i controlli, tenendo conto che i protocolli sono eseguiti per ultimi. In base alla presenza degli attributi specifici dei due motori, il sistema avvia la modalità di simulazione richiesta.

Nel progetto è stato utilizzato il motore a cicli. Il file di configurazione si occupa di includere nell'esecuzione tutti i controlli e i protocolli necessari ad essa, tra cui *Pastry*, che implementa l'interfaccia con la *DHT*, ed *Access Level* che implementa il controllo degli accessi, configurandone i parametri.

Ad esempio, la configurazione del controllo per l'*Access Level* è la seguente:

control.computeAccessLevelControl it.simulation.control.ComputeAccessLevelControl
control.computeAccessLevelControl.step 1
control.computeAccessLevelControl.frameworkOSNLevelProtocol frameworkOSNLevel
control.computeAccessLevelControl.lnk link
control.computeAccessLevelControl.churnProtocol churn
control.computeAccessLevelControl.until 2000

La prima riga definisce la classe del controllo, in seguito vengono impostati i vari parametri: la frequenza con cui opera all'interno della simulazione, un riferimento per l'associazione al protocollo per l'*Access Control* e a quello per la gestione del *churn*; inoltre è possibile impostare in quali cicli il controllo deve essere eseguito, tramite gli attributi *from* e *until*.

Nella fase iniziale della simulazione, vengono eseguiti i controlli di inizializzazione: *InitUserChurn.java*, *InitUserSocialTable.java* e *InitFrameworkOSNLevelProtocol.java*.

Il primo controllo ha il compito di impostare lo stato iniziale degli utenti. Il *churn* rappresenta la dinamicità nell'entrata e uscita degli utenti nella rete, le informazioni relative alle sessioni di attività degli utenti sono ricavate dall'applicativo *Social Circles!*. *InitUserChurn* si occupa di caricare tali informazioni nel nodo, verificare lo stato iniziale dei nodi, analizzando il file *NodeIdavailability.csv* di *Social Circles!*, per poi settare nello stato di inizio simulazione.

Il secondo controllo, *InitUserSocialTable*, é finalizzato alla ricostruzione del profilo degli utenti presenti nella rete, in particolar modo delle informazioni riguardanti le loro amicizie. Tramite la libreria *CSVReader* si aprono i file e vengono esaminate le informazioni ricavate mediante l'applicazione *Social Circles!*. Vengono creati i nodi, strutture che conterranno tutte le informazioni riguardanti il profilo utente e la relativa *social table*, la tabella composta dai dati concernenti le amicizie dell'utente. Tra le amicizie degli utenti sono presenti sia i contatti che hanno effettuato la registrazione al software *Social Circles!*, sia quelli che non hanno partecipato, di quest'ultimi non sono registrate informazioni dettagliate, se non la loro relazione di amicizia con l'utente.

L'ultimo inizializzatore, *InitFrameworkOSNLevelProtocol*, si occupa di inizializzare i protocolli, e gestisce l'inizializzazione delle politiche definite

dall'utente all'interno del *framework*. I protocolli descrivono il comportamento dei nodi ad ogni ciclo di simulazione. In ogni protocollo viene poi inizializzata un'istanza di *Balana*, che sarà necessaria nella valutazione delle richieste ai contenuti.

Il controllo, infine, fornisce metodi statici per la creazione, il caricamento da file e la stampa delle politiche definite. Tali metodi vengono usati per l'introduzione delle politiche nella simulazione.

Gli inizializzatori, come già descritto, vengono eseguiti una sola volta prima dell'avvio del motore a cicli.

4.4 Controlli

Le procedure ripetute periodicamente durante l'esecuzione dei cicli di simulazione sono i controlli.

Come descritto, nel file di configurazione è possibile impostare le tempistiche di attivazione: i controlli possono essere attivi sin dall'inizio oppure attivarsi ad un determinato ciclo ed è possibile definirne l'esecuzione ad intervalli di cicli regolari, chiamati *step*.

Nella simulazione sono presenti i seguenti controlli:

- *ComputeAccessLevelControl*,
- *GetRequestStatisticsAccessLevelControl*,
- *NumberNodesOnline*,
- *ProduceAndSearchKey*,
- *GenerateRequestControl*
- *SchedulerChurnModel*.

ComputeAccessLevelControl viene eseguito ad ogni ciclo della simulazione, si occupa di schedulare il protocollo che effettuerà la migrazione del profilo nel caso di disconnessione dell'utente. Si esaminano quali, fra i nodi online, si disconnettono nel ciclo corrente (informazione ricavata dalle sessioni legate all'applicativo *Social Circles!*): una volta trovati questi nodi viene creata la lista composta dai contatti in relazione di amicizia con ognuno di questi nodi. Ogni contatto riceve due punteggi: il primo riguarda la percentuale di profilo

dell'utente su cui ha accesso e il secondo è relativo alle durate delle sessioni online. Il punteggio indica il grado di accesso ed è calcolato simulando le richieste da parte dei nodi, relativi agli elementi contenuti nella lista, verso l'utente. Il secondo punteggio prende in esame non solo il comportamento giornaliero dei nodi, ma valuta le sessioni condivise fra gli stessi nodi e l'utente.

Viene definita una euristica, mediante la quale si scelgono 3 contatti: il nodo con il punteggio complessivo migliore, quello che garantisce le performance migliori e un nodo casuale; tali nodi sono considerati come campioni per effettuare la valutazione delle strategie di *Access Control*, descritte più in dettaglio nel Capitolo 5.

GetRequestStatisticsAccessLevelControl preleva le statistiche dal controllo appena descritto, crea un file con estensione *CSV* e, per ogni nodo, inserisce le informazioni relative ai suoi attributi. Tale controllo non viene solitamente attivato ad ogni ciclo, ma viene configurato con un intervallo significativamente ampio di cicli per limitarne il peso computazionale e migliorare la leggibilità del file *CSV* di report creato.

NumberNodesOnline è un controllo che ha il compito di enumerare i contatti online. Nel momento in cui viene eseguito calcola il numero di contatti registrati attivi e quello degli utenti non registrati online.

Il controllo *ProduceAndSearchKey* ha lo scopo di simulare il comportamento della rete *DHT* per testarne il corretto funzionamento. Ogni nodo, inizialmente, inserisce una sua chiave generata casualmente e un messaggio di *PUT*; in seguito, ad ogni successiva esecuzione del controllo, il nodo tenterà di recuperare la sua chiave con un messaggio di *GET*. Il test stabilisce quante tra le richieste abbiano ricevuto una risposta attraverso il protocollo di *Pastry*.

Il *GenerateRequestControl*, per ogni utente registrato, sceglie uno dei contatti amici ed effettua una richiesta di accesso al profilo. La scelta dell'amico non è totalmente casuale, ma influenzata dalla frequenza di interazione fra le due parti. Successivamente, vengono scelti casualmente un numero di utenti non appartenenti alla *social table* del nodo designato e si simulano delle richieste di amicizia.

Lo *SchedulerChurnModel* si occupa semplicemente di programmare l'esecuzione del protocollo relativo al modello per il *churn* per il ciclo corrente.

4.5 Protocolli

I protocolli rappresentano il comportamento dei nodi ad ogni ciclo di iterazione. Ogni nodo può eseguire più protocolli per ogni ciclo, a seconda di quanti ne sono stati definiti nel file di configurazione, in maniera sequenziale.

I protocolli possiedono un identificatore univoco tramite il quale sono accessibili. Al fine di definire un protocollo *PeerSim* richiede l'implementazione dell'interfaccia *CDProtocol* oppure *EDProtocol*, rispettivamente se la simulazione viene effettuata per mezzo di un motore a cicli od eventi, specificando il corpo del metodo *nextCycle*.

Durante l'esecuzione dei cicli, per ogni nodo, viene richiamato il metodo *nextCycle* che eseguirà il comportamento specificato. Nel sistema sono presenti tre diversi protocolli:

- *ChurnModel*,
- *FrameworkOSNLevelProtocol*.

Il *ChurnModel* è il protocollo che gestisce la dinamicità riguardante le entrate ed uscite dei nodi dal sistema. Il protocollo opera solamente nel caso in cui il nodo associato cambi il suo stato di attività. Se il nodo in un determinato ciclo deve effettuare una disconnessione, viene settato a *DOWN* il suo stato e viene aggiornata, a fini statistici, la media durata delle sue sessioni. Al contrario, se un utente effettua la connessione al servizio, il suo stato è settato a *OK* e viene incrementato il numero delle sue sessioni. Se il nodo, infine, non cambia lo stato, si incrementa la durata della sessione, sia che esso sia connesso oppure no.

Il *FrameworkOSNLevelProtocol* è il protocollo più complesso della simulazione e si occupa di generare le richieste di accesso ai profili, o parte di essi, e gestisce inoltre la ricezione delle stesse. A differenza degli altri, questo protocollo è basato sugli eventi e non sui cicli.
Le richieste *XACML* sono composte da una parte standard e da una parte parametrica che le differenzia una dalle altre. I parametri sono relativi agli identificativi del mittente, del ricevente e della parte di profilo richiesta dal mittente.

Il protocollo rappresenta il *Policy Enforcement Point* del sistema, in quanto inoltra le richieste a *Balana*. La gestione delle richieste viene gestita dal metodo *processEvent*, la ricezione può essere relativa ad una richiesta d'accesso al profilo oppure alla risposta di una valutazione del *Policy Decision*

Point. Nel primo caso la richiesta di accesso viene incapsulata in un oggetto di tipo *XACMLRequestMessage* e inoltrata al *PDP* Balana. Il risultato della valutazione a sua volta viene inserito in un oggetto di tipo *XACMLResponseMessage* e inviato nuovamente al mittente della richiesta. Nel secondo caso la risposta viene salvata in corrispondenza della richiesta iniziale. Dato che è possibile che una risposta sia ricevuta del nodo mentre è offline, essa verrà registrata all'interno della struttura come tale.

Il protocollo fornisce, poi, l'implementazione del metodo usato dai controlli per calcolare il livello di accesso di un insieme o di un singolo nodo nei confronti di un altro, rispettivamente tramite i metodi *accessLevelComponent* e *computeAccessLevel*.

Il trasferimento di informazioni fra i nodi è effettuato utilizzando il protocollo di trasporto, presente nella libreria *PeerSim*.

4.6 Il Nodo Utente

Il nodo è uno degli elementi fondamentali del simulatore *PeerSim*, in quanto gestisce la programmazione di tutti gli eventi che avvengono attraverso i protocolli ad esso assegnati.

In pratica, si tratta semplicemente del contenitore dei protocolli descritti in precedenza e di un insieme di altre informazioni utili per l'esecuzione. Alcuni tra i dati più importanti sono:

1. L'associazione tra identificatore del nodo *PeerSim* e il suo identificativo assegnato all'interno del *dataset* ottenuto da *Social Circles!*.

2. Il riferimento a tutte le informazioni del profilo dell'utente (questa parte verrà discussa più approfonditamente in seguito).

3. Variabili e procedure per la gestione dello stato del nodo nella rete e dei suoi contenuti.

Inizialmente, tutte le informazioni necessarie all'inizializzazione del nodo vengono recuperate dal file di configurazione, in particolare, poiché inizialmente l'utente non possiede informazioni da condividere, si tratta solamente di identificatori e protocolli (con i rispettivi *pId*).

Nella prima versione del sistema abbiamo considerato il profilo del nodo come piatto. Successivamente, è stato implementato il nodo in modo tale da permettere la definizione di una qualsiasi struttura gerarchica ad albero: a

tale scopo, l'elemento radice contiene riferimenti alle **macro-categorie** del profilo dell'utente, che a loro volta possono contenere ricorsivamente un numero arbitrario di altri elementi compatibili alla categoria padre.

Le macro-categorie del profilo sono:

Personal Info: contiene tutte le informazioni personali dell'utente, tra cui i dati sensibili (data e luogo di nascita, stato civile, etc.), i dati di lavoro e gli interessi.

Posts: contiene tutti i post inseriti dall'utente e da chi ha l'autorizzazione per crearne sulla sua bacheca; all'interno di ogni post sono registrati commenti e risposte ad essi.

Album: può contenere un qualsiasi numero di album annidati, ma le foglie dovranno sempre essere dei media (che possono essere immagini, video o simili).

L'introduzione del profilo strutturato ha richiesto l'uso di identificatori specifici all'interno del profilo, in modo da potersi indirizzare univocamente a tutti i contenuti. Il nuovo identificativo segue la gerarchia del profilo: la radice avrà l'id più breve e generico, i sotto-alberi manterranno la parte legata agli antenati come prefisso a cui sarà concatenato un suffisso specifico. Scendendo ai livelli più bassi dell'albero gli identificativi saranno di lunghezza maggiore, come è possibile vedere in Figura 29.

4.7 La Tabella Sociale

Durante la simulazione, tutte le informazioni relative alla rete sociale di uno specifico nodo (tra cui le relazioni di amicizia) sono mantenute in una tabella: la **Social Table**. Essa è implementata come un protocollo per i nodi, che va quindi ad aggiungersi a quelli descritti nelle precedenti sezioni.

La tabella è arricchita da due liste di nodi: quelli registrati e quelli non registrati al sistema; entrambe le liste sono strutturate come tabelle *Hash* contenenti le associazioni tra i nodi e il loro identificatori. La tabella ha una capacità iniziale fissata, che può poi essere estesa durante l'esecuzione.

Non appena la tabella è stata preparata all'esecuzione dall'inizializzatore, già descritto precedentemente, è possibile effettuare su di essa varie operazioni:

- Recuperare tutti i vicini del nodo corrente che sono anche in comune con un altro nodo dato (specificabile sia come oggetto nodo sia come identificatore),
- Conoscere il numero di amici del nodo oppure solamente quanti di questi sono online nel ciclo attuale,
- Recuperare specifici amici del nodo,
- Aggiungere un'associazione di amicizia; in tal caso, saranno impostati anche i valori relativi alla frequenza di contatto (possono essere omessi),
- Altre operazioni di ricerca e di statistiche sulle relazioni.

A supporto della *Social Table* vi è la struttura che descrive una singola entrata della tabella. Questa, non solo archivia i dati relativi alla frequenza di contatto nelle due direzioni tra i nodi, ma effettua anche il calcolo della *Tie Strength*.

4.8 Moduli Policy Information Point

I moduli per l'arricchimento delle richieste *XACML* sono associati alle *policy* definite, quindi è stata definita una classe per ognuna di esse. Ogni richiesta, per come viene generata da un nodo, richiede dell'aggiunta di attributi di cui il mittente non è a disposizione, ma che devono essere recuperati dalle strutture del nodo destinatario che sta valutando la richiesta; spesso le informazioni necessarie a rinforzare la *XACMLRequest* sono contenute nella *Social Table*.

Alcune delle politiche implementate dal sistema sono le seguenti:

- *FriendshipPolicy*,
- *CommonFriendsNumberPolicy*,
- *ActiveNetworkPolicy*,
- *Location Policy*.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    PolicyId="friendshipPolicy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit"
    Version="1"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
    http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd">
    <Description>This policy denies access to resource if accessed users has a friendship with a specified user.</Description>

    <Target/>
    <Rule Effect="Permit" RuleId="friendViewProfilePermit">
        <Description>This policy defines preferences for any friends accessing the profile of the users.</Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">readProfile</AttributeValue>
                        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                    </Match>

                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">7 Profile</AttributeValue>
                        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
                    </Match>
             <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:boolean-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#boolean">true</AttributeValue>
                        <AttributeDesignator
                            AttributeId="urn:osn:profiles:resource:friendship"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                            DataType="http://www.w3.org/2001/XMLSchema#boolean"
                            MustBePresent="true"/>
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
<!-- La condizione e' che deve esserci una relazione di amicizia con l'utente -->
        <Condition>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
                <Apply FunctionId="common-friendship-with">
                <!-- Necessario la funzione string-one-and-only perche' access-subject potrebbe essere un bags-->
                    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
                        <AttributeDesignator MustBePresent="true"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                            AttributeId="urn:oasis:names:tc:xacml:1.0:subject:sender-id"
                            DataType="http://www.w3.org/2001/XMLSchema#string"/>
                    </Apply>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
                    <AttributeDesignator MustBePresent="true"
                        Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:senderDB-id"
                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </Apply>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
                     <AttributeDesignator MustBePresent="true"
                         Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                         AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                         DataType="http://www.w3.org/2001/XMLSchema#string"/>
                 </Apply>
              <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
             </Apply>
                </Apply>
 </Apply>
        </Condition>
    </Rule>
 </Policy>
```

Figura 31. Specifica della Friendship Policy.

La prima *policy*, la cui definizione in *XACML* è riportata in Figura 31, si occupa di valutare se il nodo richiedente ha effettivamente una relazione di amicizia attiva con il nodo che effettua la valutazione. Questo è valutato mediante una richiesta di *Match* dell'attributo *friendship*, il quale non è comunicato direttamente nella *request*, e deve quindi essere inoltrata al modulo di *PIP* relativo.

Una volta ricevuta la *XACMLRequest* dal *PIP*, questo si occuperà di recuperare l'effettiva associazione dei due utenti per completare la valutazione d'accesso da parte del *PDP*. Il modulo *PIP*, inoltre, necessita dell'informazione sul mittente della richiesta, che normalmente non è specificato in una *XACMLRequest*; tale informazione è fornita al *PIP* attraverso l'arricchimento effettuato dal *PEP*, che, interfacciandosi con il richiedente, è a conoscenza della sua identità (in Figura 32 è presentato il metodo core del *PIP*, che si occupa di svolgere tale compito).

La *Condition*, in questo caso, ha lo scopo di rafforzare la sicurezza della valutazione.

```
public EvaluationResult findAttribute(URI attributeType, URI attributeId, String issuer,
        URI category, EvaluationCtx context) {
    BagAttribute result;
    // we only know about environment attributes
    if (!XACMLConstants.SUBJECT_CATEGORY.equals(category.toString())) {
        result = BagAttribute.createEmptyBag(attributeType);
    } else {
        // figure out which attribute we're looking for
        if (attributeId.toString().equals("urn:osn:profiles:resource:friendship")) {
            long start = System.nanoTime();
            try {
                EvaluationResult target = context.getAttribute(new URI("http://www.w3.org/2001/XMLSchema#string"),
                        new URI("urn:oasis:names:tc:xacml:1.0:subject:subject-id"), issuer,
                        new URI(XACMLConstants.SUBJECT_CATEGORY));
                Iterator<StringAttribute> iSubj = ((BagAttribute) target.getAttributeValue()).iterator();
                long targUser = Long.parseLong(iSubj.next().getValue());

                EvaluationResult sender = context.getAttribute(new URI("http://www.w3.org/2001/XMLSchema#string"),
                        new URI("urn:oasis:names:tc:xacml:1.0:subject:sender-id"), issuer,
                        new URI(XACMLConstants.SUBJECT_CATEGORY));
                Iterator<StringAttribute> iSend = ((BagAttribute) sender.getAttributeValue()).iterator();
                long sendUser = Long.parseLong(iSend.next().getValue());
                MyGeneralNode targNode = SocialTable.mappingDbId_NodeId.get(targUser);
                Set<AttributeValue> set = new HashSet<AttributeValue>();
                SocialTable targetNodeSocialTable = (SocialTable) targNode.getProtocol(pidSocialTable);
                if (targetNodeSocialTable.getSocialTableEntry(sendUser) != null) {
                    set.add(BooleanAttribute.getTrueInstance());
                    result = new BagAttribute(new URI("http://www.w3.org/2001/XMLSchema#boolean"), set);
                } else {
                    set.add(BooleanAttribute.getFalseInstance());
                    result = new BagAttribute(new URI("http://www.w3.org/2001/XMLSchema#boolean"), set);
                }
            } catch (Exception e) {
                result = BagAttribute.createEmptyBag(attributeType);
            }
            long end = System.nanoTime();
            averageLatency.addValue(end - start);
        } else {
            result = BagAttribute.createEmptyBag(attributeType);
        }
    }
    // if we got here, then it's an attribute that we don't know
    return new EvaluationResult(result);
}
```

Figura 32. Metodo principale del modulo *PIP* per la *FriendshipPolicy*.

La politica precedente può essere ridefinita in modo più efficiente eliminando la *Condition*, ma in tal caso si può incorrere in problemi di sicurezza. In questo caso, infatti, l'accesso può essere valutato semplicemente effettuando

il test su una variabile impostata a *true*, dopo aver specificato quale operazione si desidera fare e su quale risorsa.

Il sistema assume che tale variabile non sia configurabile dal richiedente e quindi se ne occuperà il medesimo *PIP* presentato per la *FriendshipPolicy*; tuttavia, dato che una *XACMLRequest* può essere compilata arbitrariamente inserendo tale valore nel caso in cui provenga da un'entità esterna al sistema o malevola, l'accesso potrebbe essere consentito anche ad utenti non autorizzati. La politica appena descritta non verrà considerata in seguito, a causa di questa vulnerabilità.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
PolicyId="activeNetworkPolocy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit"  Version="1">
    <Description>This policy defines preferences for any friends accessing the profile of the users.</Description>
 <!-- This Policy only applies to all requests -->
    <Target/>
       <!-- Rules
     if the rule is deemed applicable to an incoming service request, and the rule's conditions evaluate to TRUE,
     then the specified effect should be enforced
     -->
    <Rule RuleId="friendViewProfilePermit" Effect="Permit">
    <Description>This policy defines preferences for any friends accessing the profile of the users.</Description>
    <Target>
           <!--Rule Target describes the kinds of request to which a particular rule applies -->

<AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">readProfile</AttributeValue>
                        <AttributeDesignator MustBePresent="false"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                            DataType="http://www.w3.org/2001/XMLSchema#string"/>
                    </Match>
           <!--Rule Target describes the kinds of request to which a particular rule applies -->
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Alice Profile</AttributeValue>
                        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                            MustBePresent="true" DataType="http://www.w3.org/2001/XMLSchema#string"/>
                    </Match>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:boolean-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#boolean">true</AttributeValue>
                        <AttributeDesignator
                            AttributeId="urn:osn:profiles:resource:friendship"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                            DataType="http://www.w3.org/2001/XMLSchema#boolean"
                            MustBePresent="true"/>
                    </Match>
 <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:double-less-than">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#double">0</AttributeValue>
                        <AttributeDesignator AttributeId="urn:osn:profiles:resource:contactFrequency"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                            DataType="http://www.w3.org/2001/XMLSchema#double"
                           MustBePresent="true"/>
                    </Match>
                </AllOf>
             </AnyOf>
</Target>
        <!-- <Condition>
           A Condition is a predicate that must be satisfied for a rule to be assigned its effect
        </Condition>-->
    </Rule>
</Policy>
```

Figura 33. Specifica dell'ActiveNetworkPolicy.

Nella *policy* di Figura 33, oltre al consueto controllo sull'effettiva relazione di amicizia tra gli utenti, viene valutata anche la frequenza di contatto tra i nodi. È possibile impostare un qualsiasi valore di soglia l'utente desideri.

Per valutare la frequenza di contatto viene interrogato il modulo *ActiveNetworkPIPModule*, che calcola la *Tie Strength* attraverso i valori conservati nella *Social Table*, come si può vedere dalle operazioni in Figura 34.

```
// figure out which attribute we're looking for
if (attributeId.toString().equals("urn:osn:profiles:resource:contactFrequency")) {
    try {
        long start = System.nanoTime();
        EvaluationResult target = context.getAttribute(new URI("http://www.w3.org/2001/XMLSchema#string"),
                new URI("urn:oasis:names:tc:xacml:1.0:subject:subject-id"), issuer,
                new URI(XACMLConstants.SUBJECT_CATEGORY));
        Iterator<StringAttribute> iSubj = ((BagAttribute) target.getAttributeValue()).iterator();
        long targUser = Long.parseLong(iSubj.next().getValue());
        EvaluationResult sender = context.getAttribute(new URI("http://www.w3.org/2001/XMLSchema#string"),
                new URI("urn:oasis:names:tc:xacml:1.0:subject:sender-id"), issuer,
                new URI(XACMLConstants.SUBJECT_CATEGORY));
        Iterator<StringAttribute> iSend = ((BagAttribute) sender.getAttributeValue()).iterator();
        long sendUser = Long.parseLong(iSend.next().getValue());
        if (DEBUG) {
            System.out.println("SendUser: " + sendUser);
        }
        MyGeneralNode targNode = SocialTable.mappingDbId_NodeId.get(targUser);
        if (DEBUG) {
            System.out.println("Subject: dbid " + targUser + " node id->" + targNode.getID());
        }
        Set<AttributeValue> set = new HashSet<AttributeValue>();
        SocialTable targetNodeSocialTable = (SocialTable) targNode.getProtocol(pidSocialTable);
        double outContactFreq = targetNodeSocialTable.getSocialTableEntry(sendUser).getOutContactFreq();
        double MaxInteractionContact = targetNodeSocialTable.MaxInteractions();
        double tieStrength = MaxInteractionContact == 0.0 ? 0.0 : outContactFreq/MaxInteractionContact ;
        set.add(DoubleAttribute.getInstance(Double.toString(tieStrength)));
        if (DEBUG) {
            System.out.println("Out Freq: " + outContactFreq + " value: " + tieStrength);
        }
        long end = System.nanoTime();
        averageLatency.addValue(end - start);
        return new EvaluationResult(new BagAttribute(new URI("http://www.w3.org/2001/XMLSchema#double"), set));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figura 34. Parte del metodo per il calcolo della *Tie Strength*.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
PolicyId="commonFriendsNumberPolicy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit" Version="1"
xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd">
    <Description>This policy permits access to resource only if accessed users has k-common friendship relation.</Description>

    <Target/>

    <Rule Effect="Permit" RuleId="friendViewProfilePermit">
        <Description>Insert description.</Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">readProfile</AttributeValue>
                        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                    </Match>

                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">7 Profile</AttributeValue>
                        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
                    </Match>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">64</AttributeValue>
                        <AttributeDesignator AttributeId="urn:osn:profiles:resource:commonFriendsNumber"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                            DataType="http://www.w3.org/2001/XMLSchema#integer"
                            MustBePresent="true"/>
                    </Match>
                <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:boolean-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#boolean">true</AttributeValue>
                        <AttributeDesignator AttributeId="urn:osn:profiles:resource:friendship"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                            DataType="http://www.w3.org/2001/XMLSchema#boolean"
                            MustBePresent="true"/>
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
    </Rule>
</Policy>
```

Figura 35. Specifica della CommonFriendsWithPolicy.

Anche per quanto riguarda questa politica, gli attributi che dovranno essere arricchiti da un modulo *PIP*, affinché la richiesta possa essere valutata correttamente, possono essere ottenuti dalla *Social Table*. Infatti, come già descritto nella relativa sezione, la tabella è in grado di recuperare il numero di amici in comune con un dato nodo, cioè proprio quello che occorre per la valutazione dell'attributo *commonFriendsNumber*.

Il modulo di *Policy Information Point* associato a questo tipo di *policy* effettuare quindi questa operazione.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
PolicyId="LocationPolicy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit"
Version="1" xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd">
    <Description>This policy permits access to resource only if accessed users has friendship relation
    and is inside a threshold distance.</Description>

    <Target/>

    <Rule Effect="Permit" RuleId="friendViewProfilePermit">
        <Description>Insert description.</Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">readProfile</AttributeValue>
                        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
                    </Match>

                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Profile</AttributeValue>
                        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                            DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
                    </Match>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">50</AttributeValue>
                        <AttributeDesignator AttributeId="urn:osn:profiles:resource:distanceValue"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                            DataType="http://www.w3.org/2001/XMLSchema#integer"
                            MustBePresent="true"/>
                    </Match>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:boolean-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#boolean">true</AttributeValue>
                        <AttributeDesignator AttributeId="urn:osn:profiles:resource:friendship"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                            DataType="http://www.w3.org/2001/XMLSchema#boolean"
                            MustBePresent="true"/>
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
    </Rule>
</Policy>
```

Figura 36. Specifica della Location Policy.

Infine la politica *LocationPolicy* basa la valutazione dell'accesso, oltre che sulla relazione di amicizia tra i due nodi in esame, anche sulla loro distanza geografica. Per portare a termine tale valutazione, il *PDP* contatterà automaticamente il *DistancePIPModule*, il quale effettua, a questo punto, un calcolo sulla distanza geografica, mediante l'algoritmo presentato in Figura 38.

L'algoritmo utilizza la formula di Haversine [43] che calcola la **Great Circle Distance o Ortodromia**, la distanza più breve tra due punti sulla superficie di una sfera, in chilometri per poi convertirla in metri.

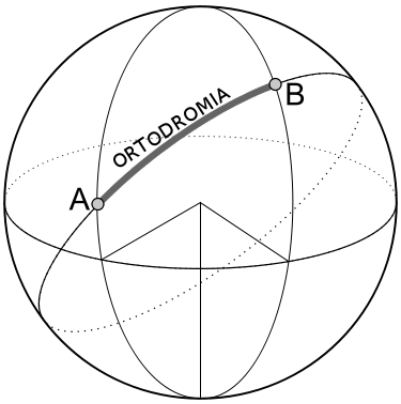


Figura 37. Esempio di distanza fra due punti in una sfera.

```
private double distance(double lat1, double lat2, double lon1, double lon2) {

    final int R = 6371; // Radius of the earth

    Double latDistance = deg2rad(lat2 - lat1);
    Double lonDistance = deg2rad(lon2 - lon1);
    Double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
            + Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2))
            * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
    Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

    return R * c * 1000;
}

private double deg2rad(double deg) {
    return (deg * Math.PI / 180.0);
}
```

Figura 38. Algoritmo per il calcolo della distanza geografica tra due nodi.

Oltre alle policy qui presentate, che sono applicabili a qualsiasi strutturazione di profilo, sono state sviluppate policy specifiche per le varie categorie gerarchiche dello *user profile*. Tra queste, alcune sono: *interestPolicy*, *MediaPolicy* (nel caso delle macro-categorie) o altre più specifiche per i singoli contenuti del profilo.

4.9 Strumenti di Supporto alla Simulazione

La classe *NodeStatistics* è una struttura associata al nodo che ne mantiene tutte le statistiche:

- *Numero di richieste ricevute*,
- *Numero di risposte offline*,
- *Numero di risposte indeterminate*,
- *Numero di accessi negati*,
- *Numero di accessi concessi*,
- *Numero di amici*,
- *Informazioni sulla latenza*.

Le classi *XACMLRequestMessage* e *XACMLResponseMessage* rappresentano le richieste e le risposte di accesso ad un dato profilo utente. Entrambe le classi contengono le informazioni relative al nodo mittente e destinatario, un identificatore univoco del messaggio ed il tempo impiegato per la sua generazione.

XACMLRequestMessage definisce il formato della richiesta *XML*, mentre nella *XACMLResponseMessage* definisce la struttura del messaggio di risposta della valutazione del *Policy Decision Point*, sempre in formato *XML*. I messaggi così strutturati vengono istanziati e ricevuti dal protocollo *FrameworkOSNLevelProtocol*, che si occupa di gestire il traffico in entrata e in uscita da ogni nodo.

Un messaggio di risposta può essere impostato a offline, nel caso in cui sia stato ricevuto in un momento in cui il nodo non era disponibile. *GET* e *PUT* sono i tipi di messaggio di interfaccia verso la *Distributed Hash Table*. In particolare, il messaggio di *GET* è una richiesta di lettura di un contenuto associato ad una chiave specificata. Nella classe *GET* è presente la chiave del contenuto da leggere, il nodo sorgente che ha richiesto la lettura di tale contenuto e il contenuto stesso.

Nello stesso modo è strutturata la classe *PUT*, che rappresenta il messaggio di inserzione di una nuova coppia (chiave - contenuto) all'interno della *DHT*. Come descritto in precedenza, i messaggi *GET* e *PUT* sono utilizzati dal controllo *ProduceAndSearchKey*.

Il sistema prevede delle classi, esterne alla simulazione, che permettono di recuperare diversi *dataset* generabili con *Social Circles!*. La connessione con il *database* in cui sono archiviati i dati sulle sessioni viene effettuata attraverso il driver *JDBC*, che consente a *Java* di effettuare *query SQL*. Tutte le informazioni così recuperate vengono inserite poi in uno o più file *CSV*; è possibile scegliere se scaricare tutto il contenuto oppure limitare i file ad una parte ridotta e, inoltre, è selezionabile anche l'insieme da cui recuperare gli id dei nodi. I file *CSV* così generati presentano un formato standard per facilitare l'esecuzione della simulazione.

Illustriamo di seguito il contenuto di alcuni tra i file più importanti:

***registeredUsersNodeIds*:** lista completa dei nodi registrati a *Social Circles!*; consiste di una sola colonna con i loro identificatori.

***GrafoOrientatoPesatoCompleteSocialCircles*:** contiene informazioni, per ogni nodo, sugli archi che descrivono le relazioni di amicizia e la frequenza di contatto registrata dallo strumento.

***RefinedOnlineStatusCompleteSocialCircles*:** in questo file sono registrati tutti i dati relativi alle sessioni degli utenti registrati: numero e loro durata media.

***usersHometownLocation* & *usersCurrentLocation*:** archiviano i dati geografici completi degli utenti. Un esempio di questa informazione è mostrato in Figura 39.

uid	city	state	country	latitude	longitude
1184858331	Sant'Agata dei Goti	Campania	Italy	41.0833	14.5
1648808774	Sant'Agata dei Goti	Campania	Italy	41.0833	14.5
1679232580	Sant'Agata dei Goti	Campania	Italy	41.0833	14.5
100000151039908	Sant'Agata dei Goti	Campania	Italy	41.0833	14.5
1320076965	Lejre	Copenhagen	Denmark	55.6	11.9833
532109807	East Kelowna	British Columbia	Canada	49.8667	-119.4
100000759078103	El Nuevo Mundo	Apure	Venezuela	7.31667	-67.4667
1181775258	Baia Mare	Maramures	Romania	47.6667	23.5833
1186237118	Baia Mare	Maramures	Romania	47.6667	23.5833
1472393980	Baia Mare	Maramures	Romania	47.6667	23.5833
100000867866661	Baia Mare	Maramures	Romania	47.6667	23.5833
100000869798067	Baia Mare	Maramures	Romania	47.6667	23.5833
100001853227210	Baia Mare	Maramures	Romania	47.6667	23.5833
100003154149566	Baia Mare	Maramures	Romania	47.6667	23.5833
1451176253	La Pointe Ã Pitre	NULL	Guadeloupe	16.2333	-61.5167
1662993059	Gallipolis	West Virginia	United States	38.7906	-82.2006

Figura 39. Frammento del dataset *usersHometownLocation*.

4.9.1 Social Circles!

Allo scopo di ottenere *dataset* significativi ed informazioni sugli utenti, in particolare per Facebook, si è utilizzato il software **Social Circles!**. Il software permette di ottenere informazioni relative alla topologia della rete, sul profilo degli utenti, sulle interazioni fra gli stessi e la durata delle loro sessioni.

L'implementazione di *Social Circles!* fa uso del sistema di *API* di Facebook, usufruibile sia da chi pubblica dati sia da chi li legge. Le informazioni collezionate sono relative ad utenti che hanno consentito il trattamento di tali dati.

Lo scopo dell'applicazione è di raccogliere dati relativi ad un buon nume-

ro di Ego Network. La caratterizzazione delle Ego Network si divide in tre tipologie di informazioni:

1. Topologia e informazioni sui profili utenti, per studiare le proprietà strutturali della rete e fornire dati per sfruttare al meglio l'algoritmo di *Social Circles!*.

2. Informazioni di interazione: analizzando post, foto, commenti, like e tag tra amici si è in grado di assegnare un "valore d'importanza" all'amicizia fra due persone. Considerando tutte le interazioni quindi è possibile dare un peso ad ogni arco di amicizia, se teniamo presente le direzioni con cui avvengono gli scambi di informazione fra i due utenti i pesi sono due.

3. Dati temporali sulla presenza online di un utente.

Tali informazioni sono utili per analizzare la co-availability, lo stato di ogni utente rispetto a quello dei suoi amici e per caratterizzare il comportamento tipico delle sessioni.

Il sistema è un portale web in cui ogni utente deve registrarsi, può accedere ed interagire con la propria ego network di Facebook. Il sito è diviso in varie sezioni, ognuna delle quali fornisce un servizio diverso:

- La pagina *Graph*
- La pagina *Statistics*
- La pagina *Interactions*
- La pagina *Friends Map*
- *About and Privacy statements*

Durante la registrazione, gli utenti devono esprimere il loro consenso al trattamento dei dati.

La pagina *Graph* mostra all'utente la propria ego network, con la quale può interagire facendo zoom e spostandone parti. Passando il mouse su un nodo della ego network si può visualizzare il nome e la foto del profilo Facebook.

La pagina delle statistiche mostra i principali interessi dell'utente e dei suoi amici, con la possibilità di visualizzare istogrammi sui film, musica e

libri maggiormente seguiti. La pagina delle interazioni fornisce un grafico degli amici più rilevanti per l'utente in base al tipo di interazioni che hanno.

La pagina *Friends map* è stata creata utilizzando le *API* di Google Map e presenta una mappa del mondo in cui i contatti amici dell'utente, sono segnalati nel luogo in cui vivono correntemente.
Le informazioni estrapolate, che hanno composto il *dataset* costruito sono le seguenti:

Topologia e informazioni sul profilo: il grafo sociale della rete di 337 ego, contenente gli utenti con i loro profili e le loro relazioni di amicizia.

Informazione di interazione: una stima della forza e della fiducia dei rapporti di amicizia, con cui è possibile la costruzione di un grafo pesato delle interazioni.

Presenza online: dati relativi al comportamento delle sessioni dell'utente.

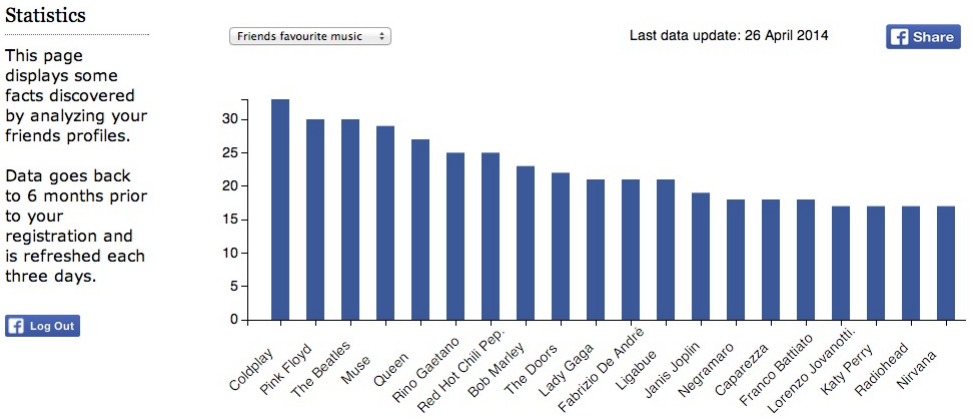


Figura 40. Pagina delle statistiche di Social Circles!.

Capitolo 5

Risultati Sperimentali

In questa sezione proponiamo una valutazione quantitativa dell'approccio proposto in questa tesi.

L'obiettivo principale è quello di determinare come sia stabilita l'*availability* dei profili e come questa sia sfruttata nella selezione di un insieme di nodi replica per ogni utente delle rete. Tale nodi replica vengono selezionati tra quelli che non hanno la necessità di ricevere un versione del profilo cifrata, in quanto sono in grado di accedere alle informazioni da trasferire. Allo scopo di poter selezionare i nodi su cui trasferire la copia del profilo, senza che essa venga crittata, occorre valutare, per ogni nodo, le sue politiche di *privacy*.

Ciò che ci interessa valutare, prima di tutto, è l'ammissibilità di tale approccio e, per fare questo, ci si è serviti di un simulatore in *Java*, descritto nel capitolo precedente, e dei dataset forniti dallo strumento *Social Circles!*. Le informazioni relative ai nodi contenute nei dataset considerati, tra cui tutte le interazioni (post, commenti, like, tag e foto), sono state recuperate nei 6 mesi precedenti alla registrazione di ogni nodo.

Per quanto riguarda il campionamento delle sessioni, *Social Circles!* traccia lo stato online/offline sulla chat di *Facebook*: le informazioni relative ad ogni utente sono state recuperate ogni 8 minuti per 10 giorni. Il dataset in totale contiene 144.481 utenti, tra cui solo 328 sono utenti registrati a *Social Circles!*, mentre gli altri sono loro amici. La popolazione così ottenuta ha il vantaggio di rappresentare un gruppo abbastanza eterogeneo di individui:

- 213 maschi,
- 115 femmine,
- età minima 15 anni,
- età massima 79 anni,
- livelli di educazione molto differenti tra loro,
- posizioni geografiche che coprono buona parte del territorio.

# Users	144,481	Avg. clustering coefficient	0.63
# Friendships	3,683,458	Avg. session number	5
Avg. degree	486.9	Avg. session length (min)	6
Avg. modularity	0.46	Avg. inter-arrival time (min)	90

Figura 41. Riassunto del contenuto del dataset.

5.1 Analisi Politiche e comportamento della Rete

Ciò che ci interessa, nei nostri esperimenti, è considerare il caso in cui è necessario assegnare una copia delle informazioni di profilo di un nodo ad un altro nodo valido come replica; questo avviene nel caso in cui il primo nodo si stia disconnettendo dalle rete *DOSN*. Per ogni nodo U che si sta disconnettendo, il sistema di elezione descritto in precedenza si occupa di selezionare dapprima tutti i nodi online che hanno una relazione di amicizia con U. A questo punto, è necessario che sia eseguita una richiesta di autorizzazione su ogni utente V nell'insieme selezionato in precedenza; la richiesta ha lo scopo di vedere se V ha i diritti di accesso al profilo di U.

Nella prima versione di *DOP*, in cui il profilo dell'utente è modellato come un oggetto unico e indivisibile, l'unico possibile esito di una valutazione è *Permit* (*Access Level* = 1) o *Deny* (*Access Level* = 0).

La durata della simulazione è uniformata con quella del periodo di scansione, in modo tale che i nodi simulino esattamente il tempo di ricerca degli utenti. Le valutazioni sulle richieste di autorizzazione sono state fatte basandosi sulle *policy* precedentemente definite e descritte nel Capitolo 4; si tratta delle seguenti politiche:

1. Politica di semplice amicizia,

2. Politica con uno specifico numero di amici in comune(2, 8 o 32),

3. Politica semplice in cui una frazione (0.02, 0.05 o 0.1) degli utenti è malevola, cioè può usare le informazioni di profilo per scopi indesiderati,

4. Politica basata sulla *Tie Strength* (0.5, 0.2 o 0.02).

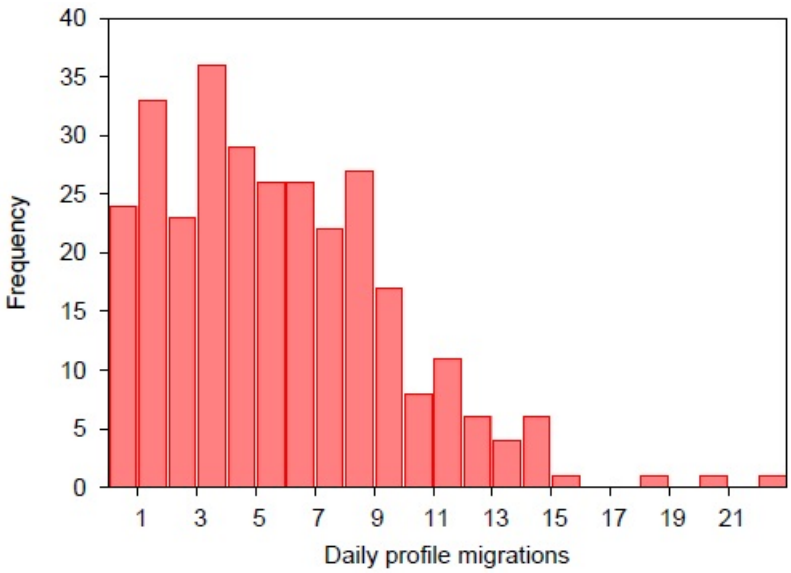


Figura 42. Grafico delle migrazioni di profilo.

In Figura 42 è illustrato il numero di migrazioni di profilo che un utente effettua in media nell'arco di una giornata; dall'analisi è emerso come questo dato sia dipendente dal numero di relazioni di amicizia.

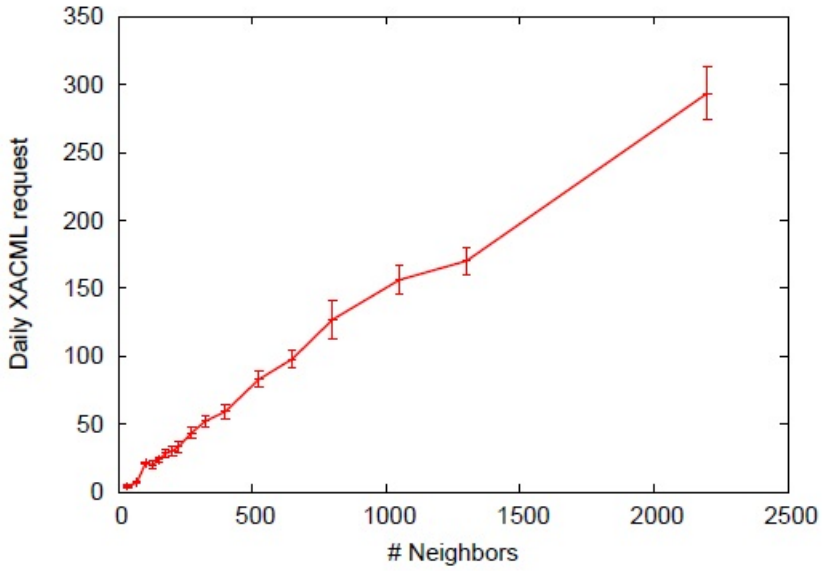


Figura 43. Numero di richieste *XACML*.

Allo stesso modo, si può vedere dalla Figura 43 come anche il numero di richieste *XACML* per l'autorizzazione d'accesso sia proporzionale al numero di amici attivi. Da questi dati è possibile capire che il comportamento della rete per quanto riguarda le richieste e le migrazioni non sia influenzato dalla *policy* specifica, ma solo dalla disponibilità del nodo e dei suoi vicini. Tra gli utenti registrati a *Social Circles!* è stato stimato che, in media, gli amici presenti online varino tra i 4 e 300.

La Figura 44, invece, mostra quanti, tra gli amici online di un nodo, siano compatibili con i requisiti imposti dalle diverse politiche (ogni linea in figura corrisponde ad una delle politiche definite in precedenza con parametri impostati in modo specifico); in questo caso, quindi, l'esito è dipendente anche dalla politica in esame.
Il numero di utenti mostrato sull'asse verticale del grafico indica il numero, in media, di vicini di un nodo che sono eleggibili come replica, senza la necessità di cifrare le informazioni di profilo.
Per quanto riguarda la prima politica esaminata, essendo i nodi selezionati già tra gli amici online dell'utente, non è necessario che sia rappresentata, in quanto coinciderebbero.

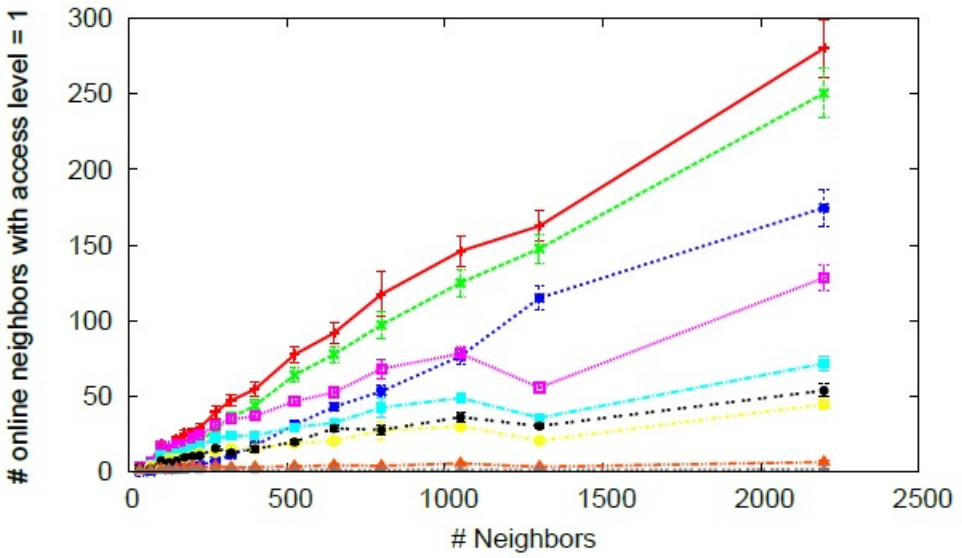


Figura 44. Numero di vicini compatibili con le *policy*.

Quanto si evince dall'analisi del comportamento dei nodi in funzione della politica di *Access Control* scelta è che gli utenti hanno una grande libertà nella scelta del livello di privacy; infatti, le tre linee che assumono valori maggiori nel grafico rappresentano la politica in esame numero 2 (tra quelle descritte a pagina 113) con parametri impostati diversamente, per poi restringere sempre di più l'accesso con le politiche 3 e 4 in ordine.

5.2 Analisi Strategie di Riallocazione

Il meccanismo di riallocazione dei dati sceglie i nodi su cui replicare i profili basandosi sul sotto insieme dei vicini connessi che, non solo soddisfano le politiche e, quindi, possono ricevere i dati non cifrati, ma anche una stima di **performance**. Per avere un riscontro reale sulle performance è stato necessario fare uso di un fattore di performance (Pf) che valutasse l'apporto parziale sui vari obiettivi richiesti.

Gli obiettivi di cui si è tenuto conto nel fattore per la valutazione delle performance sono:

1. Quante connessioni un utente ha stabilito nella rete: più relazioni possiede, più è probabile che contatti altri nodi.

2. Il numero di relazioni che due nodi della rete condividono: se un nodo ha molti amici in comune con il nodo che deve trasferire il profilo, significa che molti di quegli utenti potranno essere interessati ad accedere a quelle informazioni, facilitando le connessioni.

3. La quantità di sessioni giornaliere (numero di cambi di stato e non durata): un numero alto potrebbe significare che il profilo deve essere trasferito molte più volte, mentre un numero basso limita le riallocazioni.

4. La durata media delle sessioni: se un nodo ha periodi di connessione più lunghi aumenterà la disponibilità del profilo agli altri utenti, dato che rimane connesso al sistema per molto tempo.

Per quanto riguarda la scelta del nodo adatto ad ottenere la copia del profilo del nodo che si sta disconnettendo, si usano tre strategie:

Access Level Selection (ALS): il nodo scelto dipende dal fattore di performance *Pf*, descritto in precedenza, tra i nodi online con *Access Level* a 1 (sempre nel caso in cui il profilo sia un'unica entità).
Nel caso in cui il profilo sia struttura gerarchicamente è necessario un nuovo fattore che analizzi il *trade-off* tra *Pf* e *Access Level* determinato.

Max Performance Factor (MPF): in questo caso non si tiene in conto l'*Access Level* del nodo a cui si vuole assegnare la copia, ma si punta ad ottimizzare le performance del trasferimento, anche a costo di dover cifrare tutto il profilo (o parte di esso, nel caso sia strutturato).
Questa strategia è più adatta al caso di profilo strutturato, come vedremo in seguito.

Random Selection Strategy (RSS): un strategia usata principalmente come metro di valutazione per le altre due, dato che il nodo è scelto in modo casuale.

Queste strategie sono state valutate solamente per le *policy* descritte numero 2, 3 e 4, in quanto la *policy* 1 non restituirebbe differenti risultati applicando le prime due strategie.

Riconducendosi alla Figura 44, dove abbiamo visto come *policy* diverse possano fornire livelli di accesso differenti, valutiamo le scelta della strategia partendo dall'esito di tali politiche.

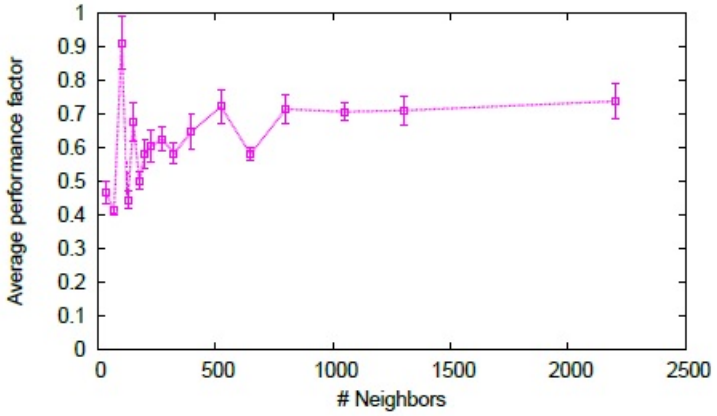


Figura 45. *Pf* valutato in base al numero di vicini per l'*MPF*.

In Figura 45 è possibile vedere come si comporta la strategia *Max Performance Factor* al variare del numero di vicini con cui è collegato il nodo che sta per disconnettersi. Essendo tale strategia indipendente dalla politica scelta, il grafico è valido per qualsiasi *policy* analizzata ed ottiene il valore di performance massimo rispetto a qualsiasi altra politica, proprio per come è definito.

Tuttavia, come si può vedere in Figura 46, i nodi selezionati da tale meccanismo risultano spesso avere *Access Level* pari a 0 (nel caso di profilo atomico) e questo porterebbe ad un elevata necessità di cifrare i profili.

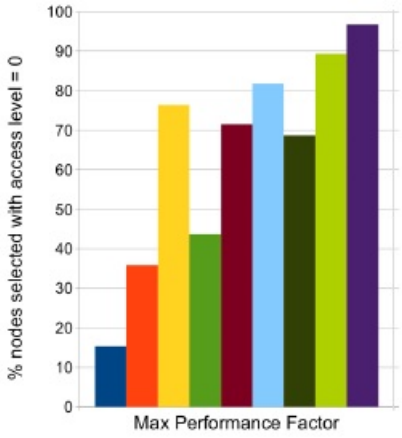


Figura 46. Nodi con *Access Level* a 0 al variare delle politiche per *MPF*.

Si può anche notare, sempre in Figura 46, che nel caso le politiche non siano particolarmente restrittive, è accettabile adottare questa strategia. In particolare, le prime colonne dell'istogramma rappresentano la politica 2 nel caso in cui gli amici comuni richiesti siano uguali o inferiori ad 8, mentre può essere ancora ancora adottato nel caso della politica 3 con 2% di utenti malevoli (colonna 4).

Poiché il nostro **requisito fondamentale è che l'*Access Level* sia accettabile** (quindi 1 nel caso di profilo unitario, altrimenti una certa soglia > 0), prenderemo la strategia *MPF* come metro di valutazione a cui vorremmo avvicinare il *Pf* della strategia *ALS*.

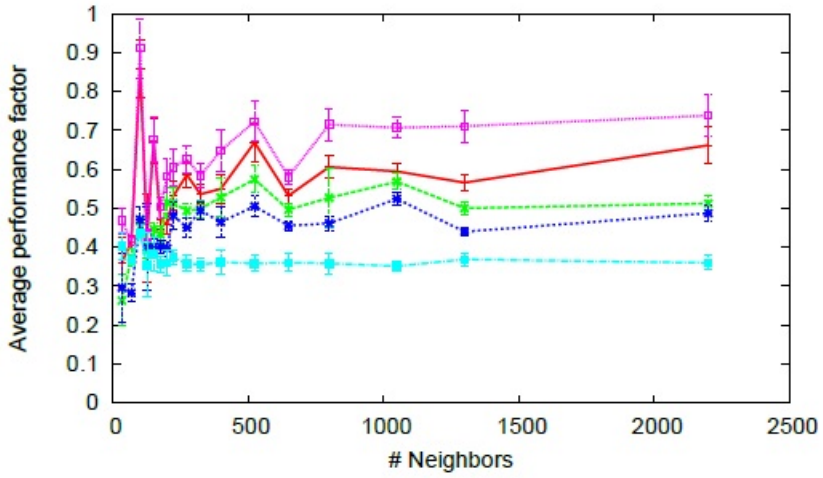


Figura 47. *Pf* valutato per tre varianti della politica 3 per *ALS*.

Prendendo in considerazioni le politiche descritte in precedenza, attraverso l'applicazione di *ALS* il *Pf* sarà sempre inferiore a quello dell'*MPF*, in quanto *upper bound*, ed in particolare andrà a decrescere all'aumentare della restrittività della politica.

Come si può vedere in Figura 47, infatti, nel caso in cui sia eseguita la politica 3, il *Pf* calerà all'aumentare del numero di vicini malevoli da escludere dal riassegnamento. In particolare, si può notare come le performance di questo approccio perdano di valore con un numero di vicini di poche centinaia, mentre siano effettivamente prossime all'*MPF* in media con politiche non eccessivamente restrittive. La linea azzurra con *Pf* più basso è l'approccio *RSS*, da cui vorremmo stare il più lontano possibile (la politica 4 spesso ci si avvicina per valori alti di *Tie Strength*).

Il motivo per il quale la strategia *ALS* sia, in definitiva, l'unica che possa essere effettivamente applicata è che il nostro obiettivo è proprio quello di evitare il più possibile la cifratura del profilo, cosa che non sarà sempre possibile o conveniente nel caso di profilo strutturato. Tutte le altre strategie, selezionando al 70% nodi che non hanno diritto di accesso al profilo dell'utente, costringeranno il nodo a crittografare i dati prima del loro invio. Infatti, *MPF* e *RSS* hanno un comportamento molto simile nella selezione di nodi, essendo tutte e due le strategie indipendenti dall'*Access Level*.

Le strategie di selezione presentate richiedono una ulteriore analisi sulla **availability** dei nodi scelti: per fare ciò è stata definita anche un fattore di *Pure Availability Measure* (*PAM*) [44]. Questa misura è definita semplicemente dalla frazione giornaliera di tempo in un cui il profilo di un dato nodo è disponibile alla rete direttamente oppure attraverso almeno una delle sue repliche. Lo scopo è ottenere che tale misura sia sempre relativamente alta, in quanto il profilo di un utente dovrebbe essere sempre accessibile al resto della rete.

La misura così definita è stata analizzata per le *policy* prese in esame in precedenza. Nel momento un cui un nodo sta andando disconnettendosi seleziona un certo numero di vicini a cui affidare una replica del suo profilo; tale numero è arbitrario, quindi effettueremo un'analisi per alcuni valori fissati per ogni strategia proposta. Per quanto riguarda la misura di *PAM* della selezione *RSS*, essa rimane invariata relativamente alla scelta della *policy* ed il valore è molto basso (da 0.7 a 0.8, in base al numero di repliche create); useremo questo valore come *Lower Bound* a cui non vogliamo mai avvicinarci. Se andiamo invece a valutare il comportamento della *MPF* scopriamo che i valori ottenuti sono molto più alti (d'altronde tra i fattori in gioco nel

calcolo del *Pf* ci sono anche durata e numero di sessioni), ma non si discostano troppo da quelli della *ALS*. Questo fatto ci fa prediligere nettamente la strategia *ALS*, dato che per l'altra i nodi sono costretti a cifrare i loro profili.

Ad esempio, se analizziamo in dettaglio la *policy* 2, troviamo che gli intervalli di *PAM* per le medesime cardinalità di repliche usate per *RSS*, sono:

MPF da 0.836 a 0.91

ALS da 0.834 a 0.90

Quindi, si discostano in media di meno di un punto, il che è trascurabile considerando che la prima codifica i profili. In particolare, generando 8 repliche, il 90% dei nodi selezionati con *ALS* ha un valori di *PAM* superiore a 0.9; questo risultato è ottimo, nel caso della *policy* 2, come si può vedere in Figura 48.

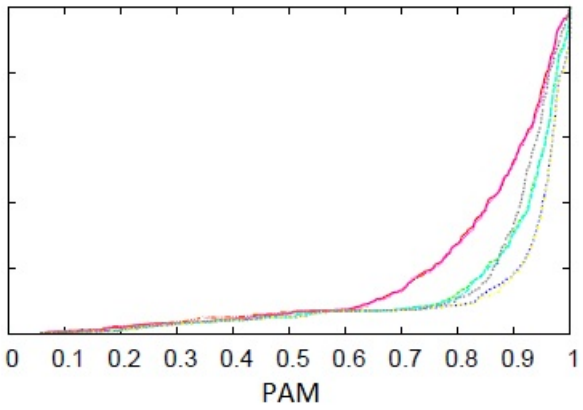


Figura 48. Crescita del *PAM* all'aumentare del numero di repliche per *ALS* e *MPF*.

Purtroppo, non si può dire lo stesso per politiche molto restrittive, come la 4, dove il valore ottenuto con *ALS* cala di quasi il 15% rispetto ad *MPF*.

Le altre politiche definite si comportano in modo simile, in base alla loro restrittività: ad esempio la politica che prende in considerazione le distanze geografiche è dipendente dal diametro scelto e la performance è inversamente proporzionale a questo valore.

Proprio per questo motivo si è considerato di strutturare il profilo in parti

e valutarne l'assegnamento basandosi sulla misura ottenuta a seguito della valutazione, tenendo in considerazione la frazione di profilo da cifrare per minimizzare il carico di computazione e allocazione aggiunto.

5.3 Analisi valutazione delle Politiche

Prima di passare all'analisi di prestazioni con profilo strutturato, dedichiamo questa sezione allo studio del tempo computazionale impegnato per la valutazione delle politiche nel sistema.

Per poter analizzare efficacemente le prestazioni di questo sistema è necessario studiarne separatamente le tre fasi:

1. Creazione della richiesta *XACML*,
2. Calcolo degli attributi necessari a completare la richiesta per la valutazione (Arrichimento dei *PIP*),
3. Tempo effettivo di valutazione della *XACMLRequest* effettuata dal *PDP Balana*.

Ciò che emerge dalla valutazione delle tre fasi è che, per ogni politica e rispettiva richiesta, il tempo di generazione della *XACMLRequest* impegna sempre pochi millisecondi ed è, quindi, trascurabile in tutte le sperimentazioni. Per quanto riguarda le altre due fasi occorre fare una considerazione basata sul numero di vicini del nodo che si vuole studiare: infatti, il comportamento del sistema risulta essere diverso al variare del numero di amici dell'utente.

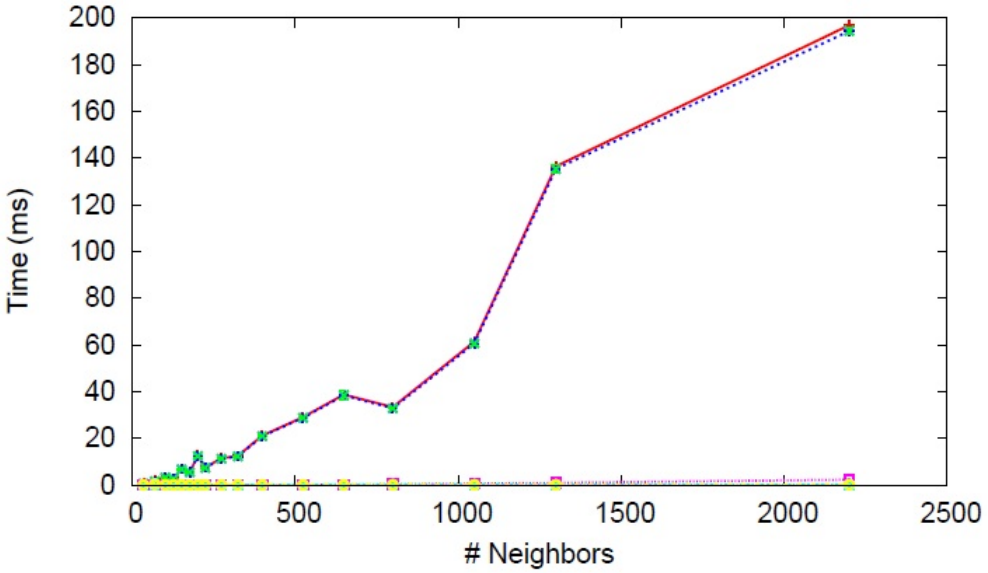


Figura 49. Tempo impiegato dalle fasi del framework di *Access Control* al variare degli amici.

Prendendo in considerazione la Politica 2, ad esempio, a cui è associato il modulo *PIP* che è adibito al recupero dell'attributo *CommonFriendsAttribute* (attributo che rappresenta il numero di vicini che sono comuni a target e richiedente), si può vedere che la fase di arricchimento, effettuata dal *Policy Information Point* per trovare i vicini condivisi, occupa quasi la totalità dell'intero processo, rendendo irrilevante la fase di valutazione (nella Figura 49, infatti, la linea tratteggiata rappresenta il tempo per la ricerca dell'attributo e quella rossa subito sopra il tempo totale, mentre le altre fasi restano prossime allo 0). Questo risultato deriva dal fatto che le operazioni necessarie al calcolo di questo specifico attributo devono considerare tutte le relazioni di amicizia dei due nodi; vedremo infatti che lo stesso non vale per le altre politiche.

Prendendo, infatti, in esame le altre tre politiche, si può notare che gli attributi da valutare per i rispettivi *PIP* sono molto più semplici, dato che consistono nel semplice calcolo di un valore su una relazione di amicizia. Questi sono, rispettivamente per le politiche 1, 3 e 4:

1. FriendshipAttribute: semplice amicizia, quindi vero o falso,

2. FriendWithAttribute: amicizia con specifici utenti, sempre vero o falso,

3. TieStrengthAttribute: valore di interazione tra i due utenti.

Tra i tre, l'unico attributo il cui calcolo risulta leggermente complesso è quello associato alla politica 3; infatti, analizzando il tempo totale di calcolo al variare dei vicini, si nota che già con 200 amici il calcolo dell'attributo supera quello della valutazione del *PDP*. Le politiche 1 e 4 hanno invece comportamento simile: fino a 500 amici il calcolo dell'attributo è meno incisivo della valutazione, oltre il tempo tende ad essere occupato, in percentuale, interamente dal recupero dell'attributo.

Il comportamento di tutte le altre politiche considerate, da quella sulla distanza geografica a quella su interessi comuni, è simile a quello delle politiche presentate (soprattutto a 1 e 4).

5.4 Analisi con Profilo Strutturato

Passiamo ora ad analizzare il comportamento del sistema nel caso in cui il profilo degli utenti sia organizzato in modo gerarchico. Attraverso questo approccio è possibile suddividere ogni profilo in parti distinte, dove ognuna di esse può essere allocata in un nodo diverso.
Dato che abbiamo definito le categorie del profilo in modo tale da abilitarne la gestione individuale, la valutazione che stiamo per proporre è fatta sulle politiche definite su tali categorie e sui relativi moduli *Policy Information Point*, dove necessario.

A questo punto occorre mettere in luce due punti molto importanti:

1. È importante notare che, per quanto le politiche definite per questa versione del sistema siano diverse da quelle descritte in precedenza, l'unica effettiva differenza ricade sul soggetto della richiesta: non avremo più politiche sul profilo intero, ma con *target subject* specifico, mentre le condizioni e le restrizioni di privacy definibili saranno le medesime.

2. A livello computazionale, per come è implementato il framework *Balana* (nello specifico il *Policy Administration Point*), il costo della prima e della terza fase delle valutazioni del *Policy Decision Point* non varia nel caso in cui quest'ultimo debba valutare una richiesta *XACML* su tutto il profilo oppure su un dato che si trova a qualsiasi livello nella struttura ad albero. Vale nuovamente l'analisi precedentemente effettuata, che

testimonia come la fase più influente sul costo computazionale della valutazione di una politica sia quella di arricchimento della richiesta da parte del *PIP*.

Possiamo stabilire che il comportamento del sistema, nel caso di semplice richiesta *XACML*, è il seguente:

- Nel caso in cui l'*Access Level* sia $\in [0, 1)$, la richiesta di accesso è stata negata.
- Se l'*Access Level* ha valore 1, esattamente come avveniva precedentemente, la richiesta di accesso è stata accettata.

Passiamo ora a descrivere il caso più delicato della riallocazione di un profilo strutturato. Come affermato in precedenza, per questa versione ha più senso definire un *Access Level* che sia diverso dalla semplice distinzione *Permit* o *Deny*: lo scopo di questo valore è quello di determinare la percentuale di profilo che un nodo è in grado di ricevere senza la necessità che venga cifrata. Se l'*Access Level* è uguale ad 1, non ci sono distinzioni con la precedente valutazione, perché il profilo è interamente replicabile su tale nodo. È però importante notare che il fattore di prestazioni di tale nodo potrebbe essere relativamente basso e tale da preferire una frammentazione su altri nodi.

Alla luce di queste considerazioni, è necessario includere, nella formula per il calcolo del *Pf*, anche il valore di *Access Level*; questo andrà ad influenzare principalmente le prestazioni della strategia *ALS*. Ciò che impatta maggiormente sulle prestazioni della valutazione di *Access Control*, per un caso di disconnessione di un nodo dalla rete, è la possibilità che vengano valutate più *policy* per uno stesso nodo; questo avviene perché, anche se siamo a conoscenza del suo *Access Level*, occorre sapere quali parti è in grado di replicare senza cifratura.

Il peso delle operazioni legate alla migrazione del profilo effettuate dal *PDP*, che nel sistema precedente erano trascurabili (veniva eseguita una sola valutazione per l'intero profilo), può avere un lieve impatto sulle prestazioni. Tuttavia, grazie al fatto che la struttura degli identificatori delle politiche è anch'essa gerarchica ed è *prefix-free*, gli accessi alle politiche sono immediati; quindi, in media, la fase di valutazione resta trascurabile, visto che il numero di valutazioni non supera quasi mai le 10 (per le politiche esaminate in precedenza).

Occorre considerare, a questo punto, il problema dell'eccessiva **frammentazione del profilo**, che porta spesso alla decisione di ridurre le parti in cui

suddividere la replica al costo di inviare una parte dei dati cifrati. Analizziamo ora più in dettaglio come si può giungere a questa decisione, mantenendo un *Pf* accettabile (non troppo distante da quello della strategia *MPF*).

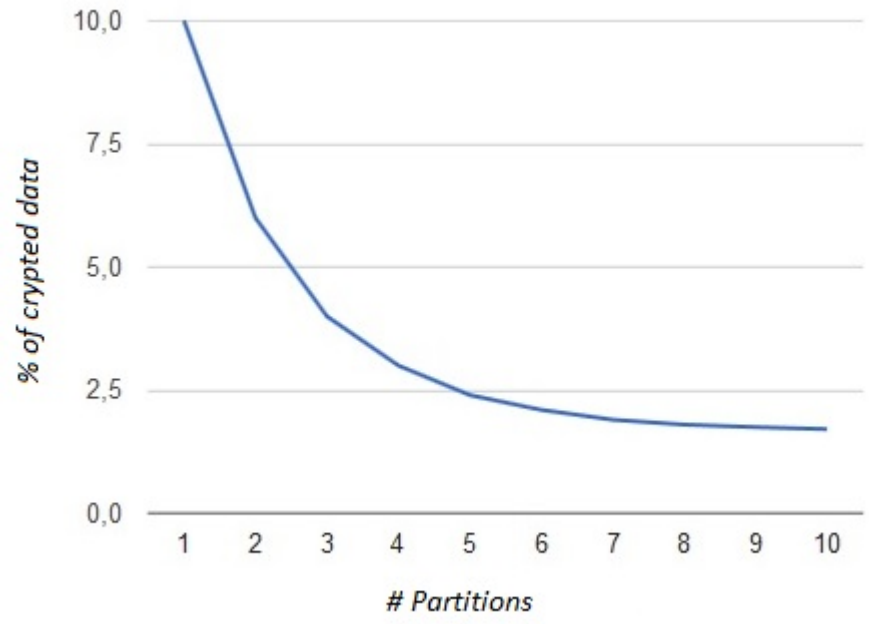


Figura 50. Analisi della variazione dei contenuti crittografati al variare del numero di partizioni.

L'analisi riportata in Figura 50 si riferisce alla relazione fra il numero di partizioni in cui sono suddivisi i profili e la percentuale dei dati che necessitano di essere cifrati. Ogni partizionamento (da 1 a 10 partizioni) è stato applicato su un numero fissato di nodi e il valore così ottenuto è una media di questi ultimi. Come si evince, all'aumentare delle partizioni in cui è suddiviso un profilo, diminuisce in maniera esponenziale la quantità di contenuti da cifrare.

L'indice delle prestazioni del riassegnamento è ora calcolato come la media degli indici dei singoli nodi a cui sono state affidate le parti del profilo. Aumentando il numero di utenti su cui effettuare la migrazione del profilo, ci si aspetta un calo delle prestazioni; mentre nella versione precedente veniva scelto il nodo compatibile con migliori prestazioni, con questa configurazione il *Pf* può solo risultare minore: infatti, sarà dato dalla media tra il nodo migliore ed altri con minori prestazioni. Inoltre, il costo computazionale del

trasferimento del profilo a più nodi aumenta all'aumentare del numero di partizioni.

Risulta quindi ragionevole limitare il numero di partizioni, poiché oltre una certa soglia il costo della migrazione dei contenuti è maggiore rispetto al vantaggio ottenuto applicando la crittografia ad una percentuale del profilo.

Capitolo 6

Conclusioni e Sviluppi Futuri

Nella tesi è stato affrontato il problema della privacy ed è stata presentata una soluzione per il controllo degli accessi per reti complesse come le *DOSNs*. Dopo avere descritto il funzionamento delle *OSNs*, si è passati ad analizzare in dettaglio le soluzioni adottate da alcune *OSN* (Facebook, Linkedin, Twitter, MySpace, Youtube) per il problema della privacy degli utenti.

Sono state poi descritte le principali tecniche, presenti in letteratura, di controllo degli accessi ai contenuti nelle *OSNs*, evidenziando i principali motivi per cui i metodi descritti non sono completamente adatti alle reti sociali.

Successivamente, sono state mostrate in dettaglio le caratteristiche delle *DOSNs* e come, in questo caso, la mancanza di entità centrali e la dinamicità elevata della rete porti ad una serie di problematiche. La gestione della privacy può essere basata su diverse tecniche, ma abbiamo messo in evidenza come, nel caso delle reti sociali, la soluzione migliore sia quella di delegare la specifica delle politiche direttamente agli utenti, i quali possono definire *policy* personali in maniera più specifica e più espressiva.

La nostra proposta è quella di implementare un controllo degli accessi in una *DOSN* basato sul linguaggio *XACML* e di gestire a livello infrastrutturale la riallocazione e la replicazione dei profili al momento della disconnessione degli utenti. Inizialmente il profilo è stato considerato come un'entità singola, sono state definite diverse politiche di accesso.

Ciò che rende innovativo il nostro approccio è la riduzione dell'uso della crittografia: si è definito il profilo come una struttura gerarchica di contenuti, la quale ha lo scopo di permettere di replicare separatamente frammenti del profilo utente principalmente sui dispositivi degli amici dell'utente. In

questo modo, poiché tali utenti avrebbero comunque accesso a quei contenuti, limitiamo la cifratura al solo caso in cui nessun amico dell'utente che si sta disconnettendo sia online.

L'analisi del sistema è stata eseguita in due fasi:

1. Sono state analizzate le prestazioni utilizzando il profilo non strutturato e si è notato che esse dipendono dalla restrittività delle politiche definite. Avere una politica più restrittiva significa avere meno scelta nell'assegnamento del profilo, che più facilmente sarà trasferito a nodi con minori prestazioni o, nel caso peggiore, i contenuti dovranno essere cifrati.

2. L'analisi con profilo strutturato ha mostrato che le prestazioni generali, in caso di migrazione del profilo a seguito di una disconnessione, migliorano applicando un numero limitato di partizioni.

In generale, il costo della valutazione delle politiche è maggiormente influenzato dall'esecuzione dei moduli *PIP*, che accedono alla struttura dati per il recupero degli attributi dell'utente.

Il sistema definito rappresenta un punto di partenza per eventuali sviluppi futuri; dato che è stato sviluppato in una simulazione, in primo luogo, andrebbe applicato e testato in una *DOSN* reale. La difficoltà di leggibilità e comprensione del linguaggio *XACML*, rende necessario sviluppare il modulo *Context Handler* come un'interfaccia *user-friendly* che renda possibile la definizione delle politiche di accesso in un linguaggio meno tecnico e facilmente utilizzabile dall'utente.

Un miglioramento da apportare, inoltre, potrebbe essere l'introduzione di un algoritmo efficiente che si occupi della delicata fase del partizionamento del profilo strutturato.

Sarà inoltre interessante utilizzare altre tecniche, magari anche con costi di esecuzione maggiori, per stimare in maniera più precisa il comportamento online degli utenti.

Infine, le politiche d'accesso che possono essere definite tramite il linguaggio *XACML* sono potenzialmente infinite, specie se applicate ad un profilo strutturato ed il livello di granularità che il sistema può raggiungere è molto elevato.

Bibliografia

[1] B. Carminati, E. Ferrari e A. Perego, "Enforcing access control in web-based social networks". In: *ACM Transactions on Informationand System Security (TISSEC) 13.1*, 2009, p. 6.

[2] B. Ali, W. Villegas e M. Maheswaran, "A trust based approach for protecting user data in social networks". In: *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research.*, 2007, pp. 288-293.

[3] S. R. Kruk, S. Grzonkowski, A. Gzella, T.Woroniecki e H.-C. Choi, D-FOAF: Distributed identity management with access rights delegation". In: *The Semantic Web-ASWC 2006. Springer*, 2006, pp. 140-154.

[4] H. C. Choi, S. R. Kruk, S. Grzonkowski, B. Davis e J. Breslin, *Trust models for community aware identity management.*, 2006.

[5] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu e B. Thuraisingham, "Semantic web-based social network access control". In: *Computers & security 30.2*, 2011, pp. 108-115.

[6] A. Tootoonchian, S. Saroiu, Y. Ganjali e A. Wolman, "Lockr: better privacy for social networks". In: *Proceedings of the 5th international conference on Emerging networking experiments and technologies. ACM.*, 2009, pp. 169-180.

[7] A. C. Squicciarini, M. Shehab e F. Paci, "Collective privacy management in social networks". In: *Proceedings of the 18th international conference on World wide web. ACM.*, 2009, pp. 521-530.

[8] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall e J. Reagle, "The platform for privacy preferences 1.0 (P3P1. 0) specification". In: *W3C recommendation 16*, 2002.

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

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

[11] A. Boutet, A.-M. Kermarrec, E. Le Merrer e A. Van Kempen, "On the Impact of Users Availability in OSNs." In: *Proceedings of the Fifth Workshop on Social Network Systems, SNS '12*, 2012, pp. 4:1–4:6.

[12] R. Narendula, T. Papaioannou e 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)*, 2012, pp. 166–175.

[13] S. Blond, F. Fessant e 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*, 2009, pp. 472–484.

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

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

[16] R. Sharma e A. Datta, "SuperNova: Super-peers Based Architecture forDecentralized Online Social Networks." In: *CoRR, abs/1105.0074*, 2011.

[17] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic e R. Steinmetz, "LifeSocial.KOM: a secure and P2P-based solution for online social

networks." In: *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, 2011, pp. 554–558.

[18] D. Liu, A. Shakimov, R. Cáceres, A. Varshavsky e 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*, 2011, pp. 61–80.

[19] J. W. Mickens e B. D. Noble, "Exploiting Availability Prediction in Distributed Systems." In: *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3, NSDI'06*, 2006, pp. 6–6.

[20] Jeffrey Travers e Stanley Milgram, "An experimental study of the small world problem." In: *Sociometry*, 1969, pp. 425–443.

[21] "Anatomy of Facebook." *https://www.facebook.com/notes/ facebook-data-team/anatomy-of-facebook/10150388519243859*, 2011.

[22] L. A. Adamic, O. Buyukko e E. Adar, "A social network caught in the web." In: *First Monday, 8(6)*, 2003.

[23] K. Barker, M. Askari, M. Banerjee, K. Ghazinour, B. Mackas, M. Majedi, S. Pun e A. Williams, "A data privacy taxonomy." In: *A. P. Sexton, editor, BNCOD, volume 5588 of LNCS*, 2009, pp. 42-54.

[24] Leanne Wu, Maryam Majedi, Kambiz Ghazinour e Ken Barker, "Analysis of Social Networking Privacy Policies." In: *University of Calgary*, 2010.

[25] A. Rowstron e P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems." In: *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[26] Lampson e W. Butler, "Protection.", In: *5th Princeton Symposium on Information Science and Systems*, 1974, pp. 18–24.

[27] R. S. Sandhu, E. J. Coyne, H. L. Feinstein e C. E. Youman, "Role-based access control models." In *IEEE Computer 29*, 1996, pp. 38–47.

[28] R. K. Thomas e R. S. Sandhu, "Conceptual foundations for a model of task-based authorizations." In: *proceedings of 7th IEEE Computer Security Foundations Workshop, Franconia*, 1994, pp. 66–79.

[29] R. K. Thomas e R. S. Sandhu, "Task-based authorization controls (TBAC): Models for active and enterprise-oriented authorization management.", In: *Database Security XI: Status and Pro-spects, T. Y. Lin and X. Qian, Eds. North-Holland*, 1997, pp. 166–181.

[30] R. K. Thomas, "Team-based Access Control (TMAC): A Primitive for Applying Role-based Ac-cess Controls in Collaborative Environments." In: *Proceedings of 2nd ACM Workshop on Role-Based Access Control, Fairfax*, 1997, pp. 13–19.

[31] A. Bullock e S. Benford, "An access control framework for multi-user collaborative environments.", In: *proceedings of the international ACM SIGGROUP conference on Supporting group work, Phoenix, Arizona*, 1999, pp. 140–149.

[32] B. Carminati, E. Ferrari e A. Perego, "Rule-based access control for social networks.", In: *On the Move to Meaningful Internet Systems*, 2006, pp. 1734–1744.

[33] J. Domingo-Ferrer, "A public-key protocol for social networks with private relationships." In: *Modeling Decisions for Artificial Intelligence, LNCS 4617*, 2007, pp. 373–379.

[34] B. Parducci e H. Lockhart, "eXtensible Access Control Markup Language (XACML) Version 3.0." In: *OASIS Standards Track Work Product*, 2013.

[35] M. Castro, M. Costa e A. Rowstron, "Performance and dependability of structured peer-to-peer overlays." In: *One Microsoft Way, Redmond*, 2003.

[36] A. Rowstron e P. Drusche, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems." In: *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[37] V. Goyal, O. Pandey, A. Sahai e B. Waters, "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data." In: *NSF, the US Army Research Office Grant No. W911NF-06-1-0316*, 2007.

[38] Y. Desmedt e Y. Frankel, "Threshold Cryptosystems." In: *University of Wisconsin-Milwaukee, Springer-Verlag*, 1998.

[39] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi e S. Niccolini, "Temporal Locality in Today's Content Caching: Why it Matters and How to Model it." In: *ACM SIGCOMM Computer Communication Review, Volume 43, Number 5*, 2013.

[40] A. Datta, S. Buchegger, L. Hung Vu, T. Strufe e K. Rzadca, *Decentralized Online Social Networks*, 2011.

[41] R. Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications." In: *Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE*, 2002.

[42] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, B. Bhattacharjee, "Measurement and Analysis of Online Social Networks." In: *ACM 978-1-59593-908-1/07/0010*, 2007.

[43] R. W. Sinnott, "Virtues of the Haversine" In: *Sky and Telescope 68 (2), 159*, 1984.

[44] M. Conti, A. De Salve, B. Guidi, F. Pitto e L. Ricci, "Trusted dynamic storage for dunbar-based p2p online social networks." In: *On the Move to Meaningful InternetSystems: OTM 2014 Conferences*, 2014, pp. 400-417.

[45] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek e R. Morris, "Designing a DHT for low latency and high throughput" In: *National Science Foundation under Cooperative Agreement No. ANI-0225660*, 2005.