UNIVERSITÀ DEGLI STUDI DI PISA

Facoltà di Scienze, Matematiche, Fisiche e Naturali
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

TESI DI LAUREA

Service Discovery in Mobile Social Networks

CANDIDATO:
Emanuele Simonelli

RELATORE:
Prof. Stefano Chessa

CONTRORELATORE:
Prof.ssa Laura Ricci

CORRELATORE:
Michele Girolami

ANNO ACCADEMICO 2013 - 2014

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

I sistemi appartenenti alla categoria delle Online Social Networks (OSNs), come Facebook, Twitter ed altri, sono sempre più utilizzati per la condivisione di informazioni e per la comunicazione diretta tra i vari nodi. In questi sistemi le comunicazioni seguono dei pattern ancora oggi in fase di studio: l'analisi delle reti sociali può essere ricondotta, attraverso un formalismo matematico, alla teoria dei grafi [Bar69] che, insieme ai risultati degli studi in campo sociale [MSLC01], fa parte di un framework di analisi definito come social network analysis [Sal12]. Attraverso questo modello è possibile analizzare i legami tra gli esseri umani, distinguendo tra incontri e relazioni che avvengono in maniera casuale, ed altri che invece derivano dalle relazioni esistenti tra le persone (legami di amicizia, di familiarità, etc).

Mobile Social Networks L'utilizzo sempre più diffuso dei dispositivi mobili (Figura 1.1), come smartphones e tablets, ha portato alla creazione di un nuovo tipo di reti "sociali" di comunicazione, evoluzione delle OSNs: le Mobile Social Networks (MSNs). In queste reti la mobilità degli utenti è un aspetto chiave per le comunicazioni, dato che queste seguono una paradigma *opportunistico* e *sociale*. Il termine "opportunistico" si riferisce al fatto che due nodi della rete creano un collegamento tra loro solo quando si trovano ad una distanza tale da stabilire una connessione attraverso le proprie interfacce wireless: la prossimità di due nodi crea quindi un'opportunità di comunicazione. Il termine "sociale" si riferisce alla possibilità di sfruttare il comportamento degli utenti per l'ottimizzazione di protocolli ed applicazioni.

Nodi di una MSN Per evitare confusione, nel resto della tesi utilizzeremo il termine *nodi* per identificare l'insieme eterogeneo di dispositivi che fanno parte di una MSN, mentre con i termini utente o persona ci riferiremo agli individui che si muovono all'interno di una MSN. Tali dispositivi, come smartphones, tablets, palmari, laptops, etc, sono in qualche modo associa-

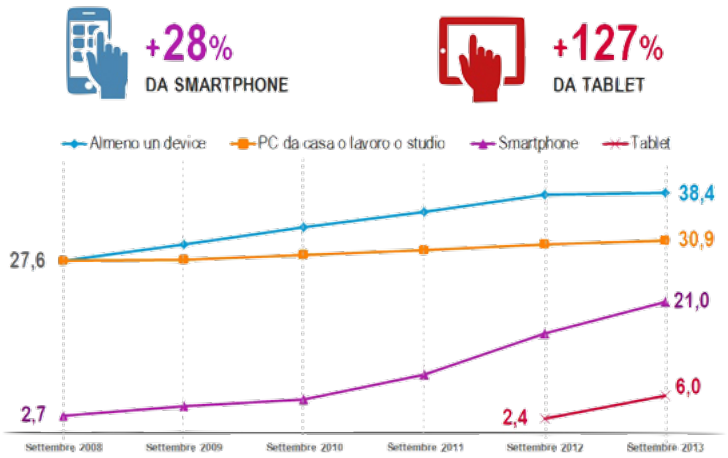


Figura 1.1: Diffusione dei dispositivi mobili

ti agli utenti, e possono essere dotati di diversi tipi di interfacce di rete, a corto raggio (WiFi, Bluetooth, ZigBee) o a lungo raggio (GPRS, UMTS, 3G, LTE); in più i nodi possono disporre di capacità di calcolo aggiuntive (CPU, GPU dedicate, memorie esterne) e di un insieme di sensori come accelerometri, giroscopi e GPS. Le risorse hardware e software descritte possono

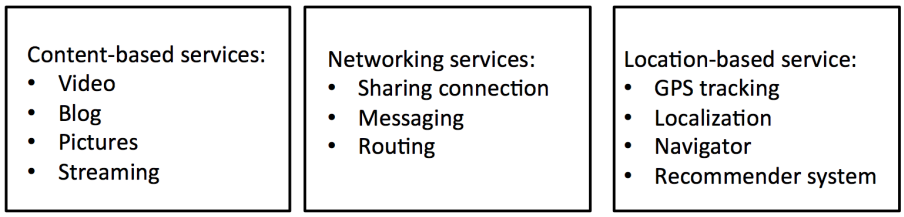


Figura 1.2: Esempi di servizi in una rete sociale

essere condivise ed utilizzate da altri nodi per offrire l'accesso a diversi tipi di *servizi*; nella tesi ci concentreremo su servizi di diverso tipo, tra cui *(i)* content-based services, utilizzati per la condivisione di contenuti multimediali, *(ii)* networking services, utilizzati per accedere alle funzionalità della rete, e *(iii)* location-based service, che offrono informazioni sulla posizione del nodo (Figura 1.2). Sulle MSNs sono stati affrontati diversi problemi noti sulle reti IP-based, tra cui routing, data dissemination, e service discovery. Quest'ultimo aspetto è quello affrontato nella tesi.

Routing su MSNs Un protocollo di routing su MSNs si occupa del trasporto di un messaggio e della scelta del percorso dalla sorgente alla destinazione, tenendo in considerazione le caratteristiche di queste reti prive di informa-

zioni sulla connettività. La soluzione basilare utilizza la tecnica del flooding (Epidemic Routing [VB00]), senza alcun genere di controllo sugli invii. Questo algoritmo ha lo svantaggio di inoltrare nella rete un eccessivo numero di messaggi; questo aspetto non si adatta bene alle caratteristiche delle MSNs, dove i nodi della rete sono dispositivi mobili, caratterizzati quindi da una limitata disponibilità energetica. In [LDS03] è introdotto PRoPHET, un protocollo di routing che sfrutta il fatto che alcuni incontri in una MSNs non seguono pattern casuali: PRoPHET utilizza un meccanismo di delivery predictabilities basato sulla probabilità di successo nella consegna di un messaggio, e pone delle regole che limitano l'inoltro di un messaggio basandosi sempre su proprietà probabilistiche. Come ultimo esempio, in [HCY11] viene presentato Bubble Rap, un algoritmo dove l'inoltro dei messaggi sfrutta la mobilità e la socialità dei nodi (si usa infatti il concetto strutturale di *centralità* e quello sociale di *comunità*): questo protocollo è il primo esempio nel quale vengono analizzate ed utilizzate in maniera più formale e precisa le proprietà delle MSNs.

Data Dissemination su MSNs Un protocollo di data dissemination su MSNs si occupa di diffondere i dati prodotti dai nodi considerando i problemi dovuti alla loro mobilità. Lo scopo di questi protocolli è quello di inviare dati utilizzando tecniche di forwarding dei messaggi, ed allo stesso tempo cercando di evitare il sovraccarico della rete. In [JX13] vengono distinte due classi di protocolli per il data dissemination in MSNs: *(i)* quelli con uno schema *publish-subscribe* e *(ii)* quelli con uno schema *social-aware*. Negli schemi di tipo **publish-subscribe** la rete inoltra i messaggi soltanto ai nodi che ne sono interessati o che ne hanno sottoscritto la ricezione, nonostante non vi sia un particolare collegamento tra i nodi produttori e quelli consumatori dei messaggi.

- PodNet project [LKM07] descrive un protocollo di wireless service podcasting che distribuisce contenuti sfruttando i contatti opportunistici. In questo protocollo tuttavia non sono utilizzate le caratteristiche sociali delle MSNs.
- SocialCast [CMMP08] è la prima proposta che utilizza le caratteristiche sociali di queste reti all'interno di un protocollo di tipo publish-subscribe. L'aspetto chiave utilizzato per questo algoritmo è l'assunzione che utenti con gli stessi interessi stiano più tempo in contatto rispetto ad altri.

Negli schemi di tipo **social-aware** vengono utilizzate le informazioni sociali in maniera massiccia: in questi protocolli sono le relazioni riscontrate nei nodi a guidare il processo di distribuzione delle informazioni. Esempi di questo tipo sono:

- ContentPlace [BCP08a] è un sistema di data dissemination che utilizza politiche di inoltro basate sul concetto di comunità: l'aspetto chiave di questo protocollo è la capacità di apprendere e comprendere informazioni sul comportamento dei nodi, in modo da migliorare la strategia di inoltro.
- PrefCast [LCC12] è un protocollo di data dissemination che ha come obiettivo principale quello di massimizzare la soddisfazione dell'utente in base all'attinenza, con i propri interessi, delle informazioni ricevute.

Service discovery Un protocollo di service discovery permette di scoprire l'insieme di servizi offerti da una rete, ed è definito come un processo che permette ai nodi di:

- pubblicizzare i propri servizi (advertisement)
- richiedere servizi offerti da altri nodi (query)
- selezionare il miglior servizio tra quelli messi a disposizione (selection)
- invocare tale servizio (access)

Il problema del service discovery nelle reti classiche è stato ampiamente studiato: le ultime soluzioni, come Jini[1] [Apa98], UPnP[2] [UPn08], SLP[3] [Sun00] e Bonjour[4] [App04], hanno necessità di un'infrastruttura basata su IP, e per questo motivo non sono utilizzabili nell'ambito delle MSNs. Più di recente sono state proposte delle soluzioni per reti ad-hoc [GBCF13, VP08] e per pervasive communication [CMP+05]. Queste soluzioni per reti wireless non tengono però in considerazione le caratteristiche tipiche delle MSNs. In particolar modo non considerano diverse tipologie di invio a seconda del messaggio scambiato, che sia un advertisement, una query, un messaggio di accesso al servizio o uno si risposta.

1.1 Motivazioni

Il lavoro svolto in questa tesi si è focalizzato sulla progettazione e lo sviluppo di un protocollo di service discovery su MSNs. Il processo di sviluppo dell'elaborato finale è iniziato con l'analisi dell'algoritmo SIDEMAN (ServIce

[1] Sun Microsystems

[2] UPnP Forum Steering Committee members: Broadcom Corporation, Cable Television Laboratories, Inc., Intel Corporation, LG Electronics, Microsoft Corporation, Motorola, Inc., Nokia Corporation, Panasonic, Philips Consumer Electronics, Pioneer Research, Ricoh Company, Ltd., Samsung Electronics Company, Ltd., Siemens AG, Sony Corporation, and Thomson Inc. 4

[3] Internet Engineering Task Force (IETF) standard.

[4] Apple Inc.

DiscovEry in Mobile sociAl Networks) descritto in [GBFC14], fino ad arrivare alla definizione dell'algoritmo SIDEMAN$_{PQ}$ (SIDEMAN with Pending Queries).

SIDEMAN non considera alcuni aspetti tipici delle MSNs:

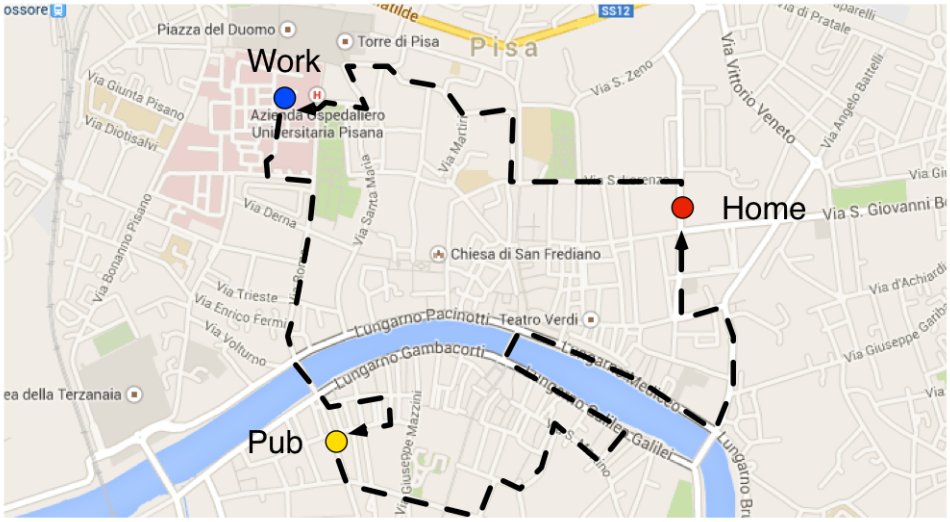


Figura 1.3: Ciclicità della giornata di un individuo

- *incontri ciclici*: i nodi seguono dei percorsi ciclici, che si ripetono nel tempo (Figura 1.3). In particolare possiamo caratterizzare la mobilità dei nodi in tre diversi aspetti chiave: *(i)* le attività comuni (ad esempio, tutti gli impiegati vanno in ufficio, tornano a casa), *(ii)* il numero limitato di luoghi visitati (per esempio, l'abitazione, l'ufficio, il cinema), *(iii)* la scelta del percorso più breve verso la meta (per esempio, per tornare a casa dall'ufficio si cerca di percorrere solitamente il percorso più breve).

- il concetto di *"friend-of-friends"*: consideriamo due nodi, n_i ed n_j, con i propri insiemi di interessi, $\mathcal{I}_i$ ed $\mathcal{I}_j$, ed i rispettivi insiemi dei nodi più frequentati, $\mathcal{C}_i$ ed $\mathcal{C}_j$; i due nodi non condividono alcun interesse ($\mathcal{I}_i \cap \mathcal{I}_i = \varnothing$), ma all'interno di $\mathcal{C}_j$ potrebbe esistere un sottoinsieme che condivide interessi con n_i. Questo aspetto potrebbe essere considerato nella fase di inltro dei messaggio (Figura 1.4): se n_i dovesse scegliere tra un insieme di nodi suoi vicini a chi inoltrare il messaggio, n_j in questo caso potrebbe rientrare nella lista dei candidati, data la possibilità di ricevere una risposta da nodi come n_k.

- *controllo della diffusione dei messaggi*: ogni volta che un nodo deve inviare un messaggio non esegue flooding sui nodi in contatto, ma

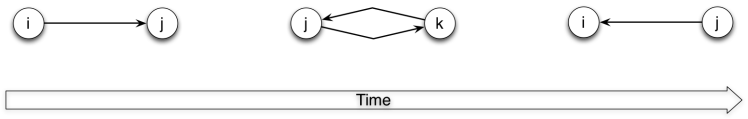


Figura 1.4: Utilizzo del concetto di *friend-of-friends*

applica delle tecniche di controllo del numero di messaggi in uscita, in base anche alle statistiche di inoltro (rate di risposta, velocità di risposta) raccolte fino a quel momento, in modo da evitare invii multipli che sovraccaricherebbero la rete.

1.2 Obiettivi

La tesi ha avuto quattro obiettivi principali:

1. l'analisi delle MSNs: questo tipo di analisi si è svolta su tracce di mobilità reali[5] raccolte da vari enti di ricerca per collezionare dati sulla mobilità umana. Tale analisi ha permesso di studiare gli scenari di riferimento per l'implementazione di SIDEMAN$_{PQ}$

2. la progettazione di SIDEMAN$_{PQ}$: nella fase di progettazione dell'algoritmo sono state utilizzate le caratteristiche mostrate nel paragrafo 1.1 insieme ai risultati ottenuti dalla fase precedente di analisi delle tracce.

3. lo sviluppo: in questa fase è stato sviluppato il codice per l'esecuzione di test nel simulatore ONE[6] [KKO10], un simulatore di reti opportunistiche.

4. la valutazione: in questa fase sono stati eseguiti i test dell'algoritmo utilizzando le tracce di mobilità ritenute più idonee. Lo scopo di questa fase è stato quello di valutare l'effettiva efficacia del protocollo sviluppato, considerando delle metriche che descrivessero opportunamente l'andamento di SIDEMAN$_{PQ}$.

1.3 Struttura della tesi

Il resto della tesi è strutturato come segue:

- nel capitolo 2 saranno descritti nel dettaglio il problema del service discovery, le MSNs con le loro caratteristiche. Saranno discusse infine le soluzioni presenti al problema del service discovery nelle MSNs.

[5]Queste e molte altre tracce di mobilità sono reperibili nella repository online CRAWDAD: http://crawdad.cs.dartmouth.edu

[6]http://www.netlab.tkk.fi/tutkimus/dtn/theone/

- nel capitolo 3 saranno descritti l'algoritmo SIDEMAN, la sua evoluzione, SIDEMAN$_{PQ}$, e gli algoritmi di community detection incontrati nella fase di progettazione degli algoritmi.
- nel capitolo 4 saranno presentate l'analisi delle tracce di mobilità, la fase di implementazione di SIDEMAN$_{PQ}$ all'interno del simulatore ONE, ed i risultati dell'algoritmo.
- infine, nel capitolo 5 sarà presentata una panoramica del lavoro svolto ed i possibli sviluppi futuri del lavoro svolto.

Capitolo 2

Concetti preliminari e stato dell'arte

In questo capitolo affronteremo il problema del service discovery ed illustreremo lo scenario delle MSNs, evidenziandone le caratteristiche principali utilizzate nei due algoritmi che analizzeremo (SIDEMAN e SIDEMAN$_{PQ}$). Infine mostreremo lo stato dell'arte delle tecniche di service discovery in scenari mobili ed opportunistici.

2.1 Service Discovery

I protocolli di service discovery hanno come obiettivo la condivisione di risorse (hardware e software) all'interno di una rete. Il service discovery è un tipico problema applicativo delle *Service Oriented Architectures* (SOA). Possiamo distinguere tre diversi tipi di nodi in una SOA:

- **service provider:** un nodo che agisce da service provider mette a disposizione degli altri nodi i suoi servizi, inviandoli all'interno della rete;
- **service client:** un nodo che agisce da service client invece non offre alcun servizio, ma al contrario invia delle richieste nella rete per poter accedere ad un servizio messo a disposizione da un service provider;
- **service directory (o service registry):** un nodo che agisce da service directory si occupa di memorizzare e gestire informazioni sui servizi disponibili all'interno della rete.

Un protocollo di service discovery permette:

- ai serivice provider di pubblicizzare i propri servizi (advertising),
- ai service client di richiedere un determinato servizio (querying),

- al service client di scegliere il servizio migliore tra quelli a disposizione (selection),
- al service client di inviare una richiesta di accesso al servizio (access).

Prima di affrontare il funzionamento dei protocolli di service discovery analizziamo le possibli modalità di invio dei messaggi utilizzabili in questo contesto.

2.1.1 Tipologie di service discovery

In un protocollo di service discovery l'obiettivo è quello di condividere risorse: per fare questo è necessario utilizzare dei meccanismi di diffusione dell'informazione attraverso l'invio di messaggi nella rete. Possiamo distinguere tra due tipologie di invio: quella di tipo *reattiva* (reactive mode) e quella di tipo *proattiva* (proactive mode).

2.1.1.1 Reactive mode

Nella tipologia reattiva (Figura 2.1a) un service client effettua una richiesta per il servizio da lui richiesto e la inoltra nella rete: in questo caso il protocollo di service discovery reagisce ad una richiesta.

Nei protocolli di service discovery sono stati utilizzati diversi meccanismi per tentare di limitare il numero di messaggi da inviare, in modo da non sovraccaricare la rete: in alcuni protocolli, per esempio, è utilizzato un time-to-leave (TTL) dei messaggi; altri protocolli utilizzano tecniche di selezione dei destinatari (selectively forward), insieme ai meccanismi di invio unicast/broadcast/multicast.

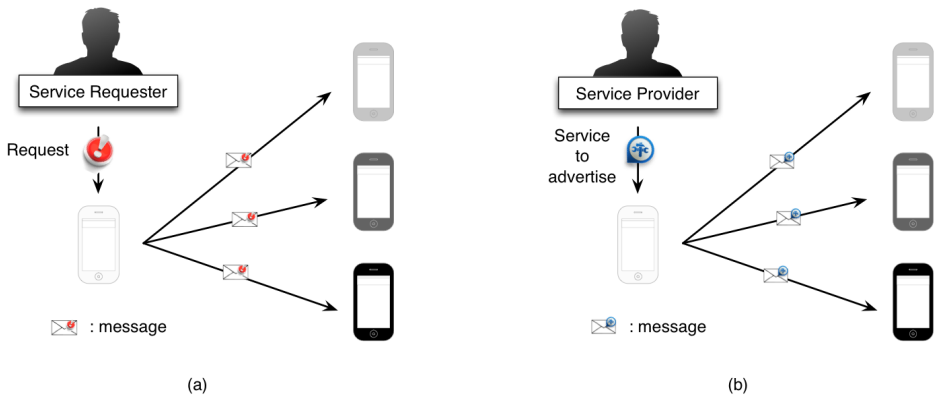


Figura 2.1: Reactive e proactive service discovery

2.1.1.2 Proactive mode

Nella tipologia proattiva (Figura 2.1b) un service provider informa gli altri nodi della rete dell'esistenza dei servizi messi da lui a disposizione, inoltrando nella rete messaggi di pubblicizzazione (gli advertisement): questi messaggi non sono il servizio, ma contengono informazioni che lo riguardano, come l'identificativo del provider che lo fornisce, una descrizione del servizio offerto, le performance ottenute utilzzando tale provider, etc.

Un possibile modo per controllare il numero di messaggi inoltrati nella rete, in questo caso, è quello di utilizzare un invio temporizzato: i messaggi di pubblicizzazione vengono inviati utilizzando un *sample time*, statico o dinamico, che indica l'intervallo tra due invii.

2.1.1.3 Hybrid mode

L'ultima tipologia è quella ibrida: possiamo ad esempio pensare ad un protocollo in cui i messaggi di pubblicizzazione dei servizi sono inviati dai service provider in maniera proattiva, ed allo stesso tempo i service client inviano richieste di accesso ai servizi in maniera reattiva.

SIDEMAN e SIDEMAN$_{PQ}$ utilizzano una modalità ibrida, come vedremo nel capitolo 3: questa strategia è stata utilizzata far accedere i service client ad un servizio nel minor tempo possibile.

2.1.2 Funzionamento dei protocolli per service discovery

Analiziamo adesso le diverse fasi di un protocollo di service discovery [VP08]. Come abbiamo già detto, in generale vi sono quattro step che compongono questi protocolli: la fase di advertisement, la fase di querying, la fase di selezione e quella di accesso, come mostrato in Figura 2.2.

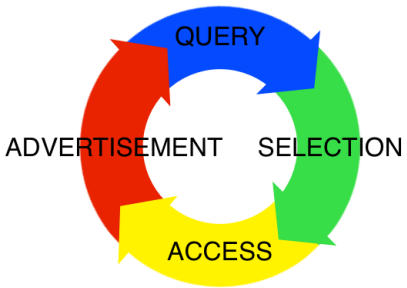


Figura 2.2: Processo di service discovery

2.1.2.1 Advertising

La fase di advertising è utilizzata dai protocolli di service discovery per pubblicizzare i servizi presenti nella rete (nel caso di SIDEMAN e SIDEMAN$_{PQ}$, servizi offerti da dispositivi mobili), ed ha l'obiettivo di informare e mantenere aggiornati gli utenti sull'insieme di servizi messi a disposizione dai provider. Un advertisement è una struttura dati compatta che descrive le

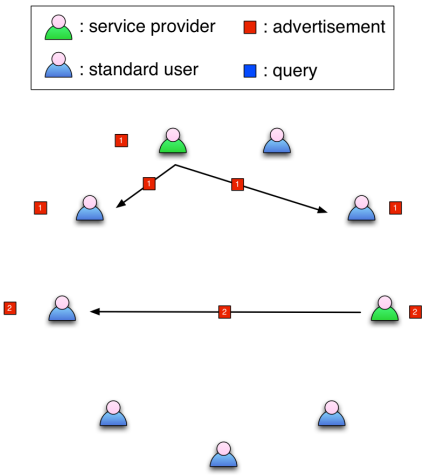


Figura 2.3: Fase di pubblicizzazione

caratteristiche più importanti di un servizio, come le sue proprietà funzionali e non funzionali garantite. Nella tesi assumeremo che un advertisement adv_j contenga almeno i seguenti campi:

- i topics del servizio: $\mathcal{I}_j \subseteq \mathcal{I}$, l'insieme degli interessi (topics) che riguardano il servizio descritto da adv_j, dove $\mathcal{I} = \{t_1, t_2, \ldots, t_n\}$ è l'insieme di tutti i possibili topics.
- il service provider: sp_j identifica il nodo provider del servizio descritto con l'advertisement adv_j.
- un timestamp: T_j è il tempo in cui è stato inviato un advertisement.
- altre informazioni aggiuntive, relative alla qualità ed alle performance del servizio descritto da adv_j.

L'insieme degli advertisement di un nodo n_i sono memorizzati in una cache $\mathscr{A}_i = \{adv_1, \ldots, adv_n\}$. Ogni volta che n_i riceve un advertisement adv_k aggiorna la propria cache,

$$\mathscr{A}_i = \mathscr{A}_i \cup \{adv_k\}$$

In un **modello proattivo** i service providers eseguono una fase di pubblicizzazione all'interno della rete per informare i service clients dei servizi da loro offerti (Figura 2.3); tale fase avviene senza che i nodi che ricevono questi advertisement abbiano fatto precedenti richieste. Invece, in un **modello reattivo**, il nodo n_i che ha degli advertisement memorizzati in cache, ovvero t.c. $\mathscr{A}_i \neq \varnothing$, li invia su richiesta quando un nodo n_j che li richiede esplicitamente, ad esempio come risposta ad una query (Figura 2.4b).

2.1.2.2 Querying

La fase di querying è utilizzata per ricercare uno specifico servizio: viene creata una query che descriva il tipo di servizio richiesto dal service client, e viene inviata nella rete. In un **modello reattivo** (Figura 2.4a) l'uten-

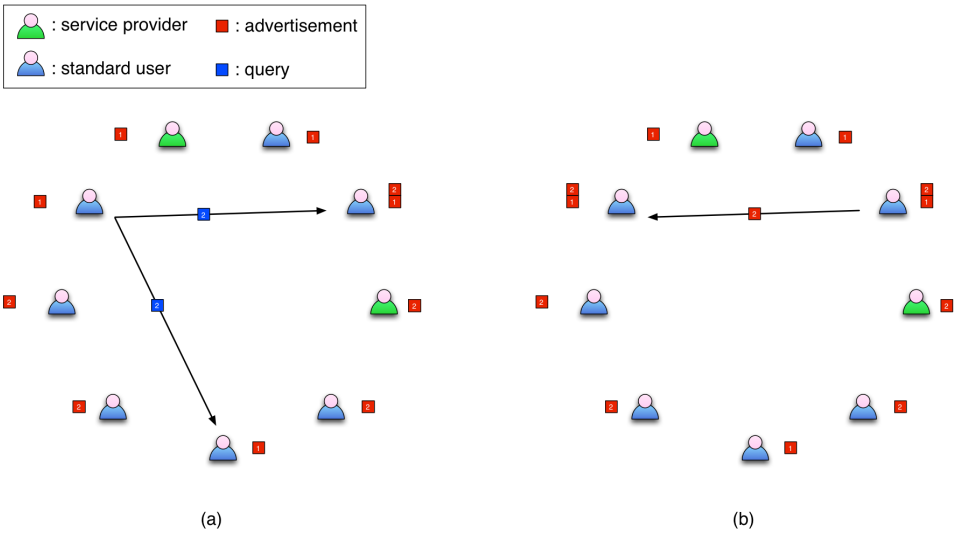


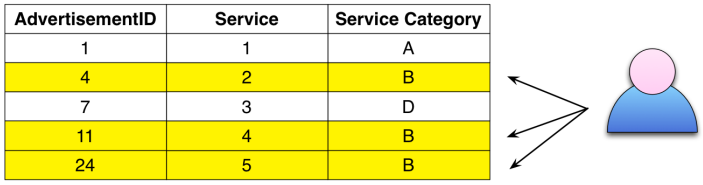
Figura 2.4: Fase di querying

te genera una query e controlla se nella sua cache $\mathscr{A}_i$ sono presenti degli advertisement che corrispondono al servizio richiesto: se c'è una o più corrispondenze, allora il nodo accede al servizio, altrimenti inoltra la query nella rete, in modo da ottenere una risposta.

2.1.2.3 Selection and access

La fase di **selezione** permette di selezionare il miglior advertisement, che descrive un servizio richiesto, presente nella cache di un nodo: la selezione è svolta in base ad una specifica funzione obiettivo definita dal protocollo. Come esempio, possiamo considerare il caso in cui i service clients abbiano l'obiettivo di selezionare l'advertisement il cui provider minimizzi il consumo di energia, oppure il tempo di risposta alla richiesta di accesso al servizio.

Dati $\mathcal{Q}$, l'insieme delle query, e $\mathcal{A}$, quello degli advertisement, modelliamo



AdvertisementID	Service	Service Category
1	1	A
4	2	B
7	3	D
11	4	B
24	5	B

Figura 2.5: Fase di selezione

la fase di selezione attraverso una funzione $\lambda : \mathcal{Q} \to \mathcal{A}$ che, data una query $q \in \mathcal{Q}$, seleziona l'advertisement adv_j che massimizza una funzione di utilità $u : \mathcal{A}_i \to \mathbb{R}$:

$$\lambda(q) = adv_j = \max_{\forall adv_k \in \mathscr{A}_i} u(adv_k) \tag{2.1}$$

Una volta selezionato l'advertisement "migliore" si passa alla fase di **accesso**, in cui il service client richiede l'accesso al service provider: in questo caso l'obiettivo principale è quello di selezionare il nodo a cui inoltrare la richiesta, sia nel caso di *service request* (Figura 2.6a), sia nel caso di *service response* (Figura 2.6b). È utilizzata, per questa fase di selezione, una funzione di accesso $\mu : \mathcal{M} \to \mathcal{V}$, dove $\mathcal{M}$ è l'insieme dei messaggi (service request/response) e $\mathcal{V}$ è l'insieme dei nodi. La funzione di accesso, dato un messaggio $m \in \mathcal{M}$, restituisce il nodo a cui inoltrare m.

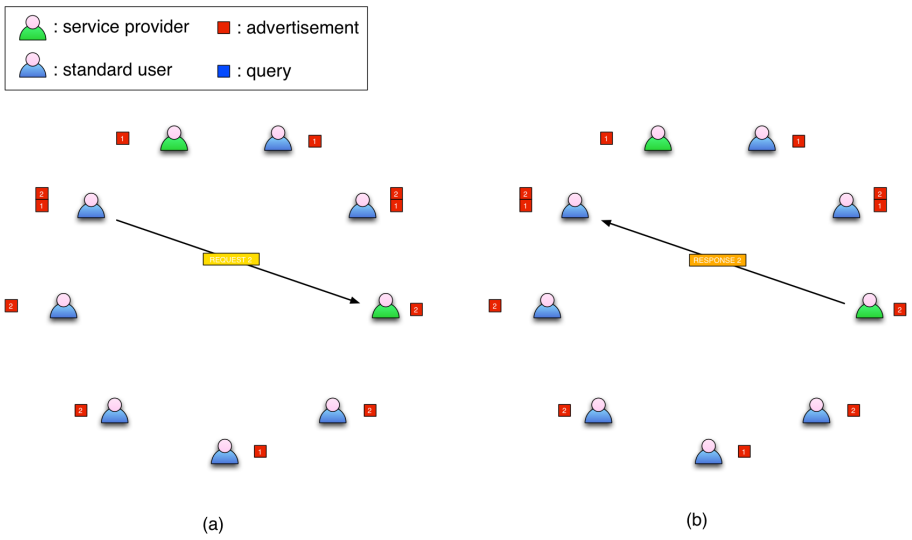


Figura 2.6: Fase di accesso

Ci sono due casi per l'invio del messaggio m:

- se il service client è in contatto con il service provider, allora m è inviato direttamente,
- se il service client non è in contatto con il service provider, allora m è inoltrato ad un nodo n_j ritenuto "migliore".

2.1.3 Architetture per service discovery

Le architetture per service discovery si distinguono in tre famiglie [VP08]: directory-based, directory-less ed ibride. Queste a loro volta possono essere centralizzate, distribuite oppure ibride.

2.1.3.1 Architetture directory-based

Nelle architetture directory-based un service provider registra i propri servizi presso uno o più service repository, ed un service client riceve informazioni sui servizi disponibili nella rete attraverso i service repository. Un esempio di questa tipologia di architettura è quella mostrata in Figura 2.7: in questo

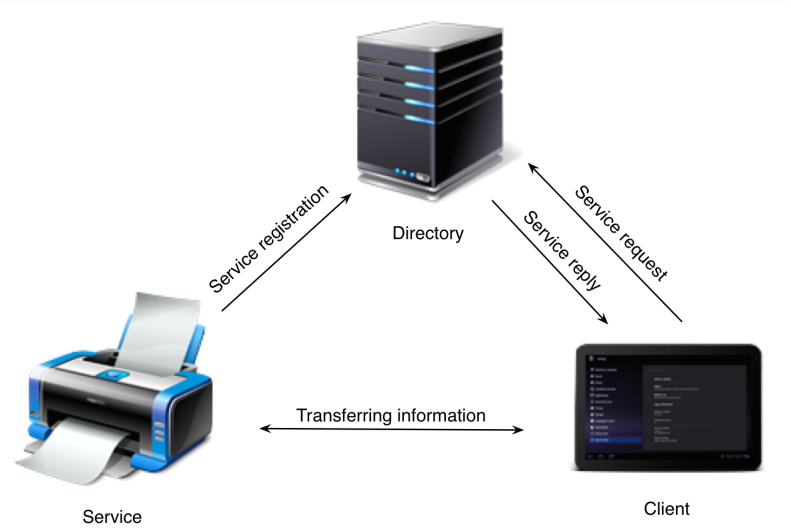


Figura 2.7: Directory-based architectures

esempio abbiamo una stampante (il service provider) che registra i propri servizi presso una service directory (in questo caso un server - struttura centralizzata); l'utente attraverso un tablet (il service client) vuole usufruire del servizio, quindi esegue una richiesta alla directory, che lo indirizzerà verso la stampante.

Una directory può essere implementata in maniera centralizzata o distribuita: nell'esempio appena visto la directory è centralizzata, ma possiamo immaginare anche una situazione analoga in cui l'accesso è eseguito su una

directory distribuita, ad esempio su cloud. La directory centralizzata non è una buona soluzione sotto diversi punti di vista:

- *single point-of-failure*: in caso di errori o impossibilità nell'accesso alla directory il servizio non sarebbe disponibile;
- *scalabilità*: i messaggi di registrazione del servizio da parte dei service providers, ed i messaggi di richiesta di accesso al servizio da parte dei service clients sarebbero destinati tutti alla solita directory, creando un *bottleneck*.

Per queste motivazioni le directory distribuite sono un approccio preferibile per questo tipo di protocolli.

Utilizzando un approccio distribuito è possibile utilizzare diverse tecniche per garantire la consistenza dei dati tra le varie service directories e l'accesso globale ai servizi:

- *full-replication* delle directory: utilizzata per esempio in Jini, dove un piccolo numero di nodi, chiamati lookup servers, vengono utilizzati come directory. In questo approccio non abbiamo comunicazione tra i componenti della directory distribuita;
- costante comunicazione tra le directories;
- *distribuited hash table*: utilizzate con l'aggiunta di informazioni di localizzazione.

Un'aspetto che penalizza questo tipo di architetture è il costo addizionale delle comunicazioni per il mantenimento della struttura delle directory e per la consistenza delle informazioni sui servizi distribuite tra le varie directory. In ambito mobile è necessario tenere in considerazione anche il fatto che questi dispositivi hanno una durata limitata, soprattutto se esposti all'invio di un gran numero di messaggi. Sarà importante tenere in considerazione questo aspetto per tutto il resto della trattazione.

2.1.3.2 Architetture directory-less

Nelle architetture directory-less gli advertisement dei servizi presenti nella rete sono distribuiti nella rete e non si trovano, come nel caso precedente, localizzati tutti in una directory. In questo caso sarà quindi necessario definire delle strategie per l'inoltro dei messaggi (*forwarding*) provenienti da service clients e da service providers:

- **broadcasting:** problemi legati alle repliche, e quindi al numero di messaggi, che si trovano nella rete;
- **scheduling and prioritization:** i server provider periodicamente inviano in broadcast gli advertisement esclusivamente ai vicini (one-hop neighbors);

- **multicasting:** al posto del broadcasting, in modo da bilanciare il carico sulla rete;
- **selective/probabilistic forwarding:** si utilizza il valore della probabilità di ricezione del messaggio per valutare i migliori candidati per l'invio di un messaggio.

È possibile anche utilizzare soluzioni architetturali ibride: per esempio, se un service provider si trova nelle "vicinanze" di una service directory, allora registra li il suo servizio, altrimenti lo invia in broadcast (o con le altre modalità che abbiamo appena visto); la stessa cosa fanno i service client, richiedendo ad una service directory se presente, effettuando il broadcast altrimenti.

2.2 Mobile Social Networks

Le mobile social networks (MSNs) [VY13, Hum10] sono un'evoluzione e specializzazione delle online social networks (OSNs), come Facebook, Twitter, LinkedIn, etc. Le MSNs sono una tipologia di reti che appartiene alla categoria delle delay-tolerant networks (DTNs), caratterizzate da latenze di comunicazione molto grandi. Le MSNs hanno ereditato, dalle DTNs, il paradigma di comunicazione *store-and-forward*: un nodo n_i che riceve un messaggio m destinato ad un nodo n_j, con il quale però non è in contatto, memorizza m in attesa di incontrare n_j e recapitargli tale messaggio (anche questo aspetto è legato alla forte dinamicità di questo tipo di reti). Le MSNs sono studiate sotto diversi punti di vista, tra cui *(i)* quello dell'informatica e delle scienze della comunicazione in generale, nelle quali si analizzano i protocolli e le architetture presenti, e *(ii)* quello delle scienze sociologiche, in cui si analizza il comportamento dell'individuo all'interno di uno o più gruppi di persone ed i modelli di mobilità umana. Queste reti hanno alcune interessanti proprietà (descritte nel paragrafo 2.2.2) che le differenziano rispetto alle altre forme di reti (ad esempio le WSN, le MANET, etc). Lo studio e l'analisi delle MSNs ha portato alla luce alcuni aspetti importanti da sfruttare per la realizzazione di protocolli efficienti all'interno di queste reti.

Le reti sociali possono essere definite come *user-centric mobile communication systems*: la possiblità di un utente di scambiare informazioni con gli altri membri di una MSN aumenta con l'utilizzo di algoritmi che utilizzano le proprietà sociali di queste reti.

Nei prossimi paragrafi saranno presentate le possibili infrastrutture (2.2.1) e gli aspetti sociali (2.2.2) legati alle MSNs.

2.2.1 Architetture e componenti di una MSN

Le MSNs si sono evolute negli anni, utilizzando architetture sempre più complesse e distribuite. Tale evoluzione è stata fortemente condizionata da una

grande diffusione di smart devices come smartphone, tablet, smartwatch, etc.

Possiamo identificare tre diverse architetture per le MSNs: centralizzata, distribuita ed ibrida.

2.2.1.1 MSNs centralizzate

L'utilizzo di una struttura centralizzata (Figura 2.8) ha diverse caratteristiche:

- tutte le informazioni relative agli utenti di una MSN sono memorizzate all'interno di un server centrale;
- i servizi offerti dalla rete sono generalmente di buona qualità;
- tutte le comunicazioni passano dai gateway, e questo aspetto introduce dei problemi:
  1. si presentano bottlenecks presso questi dispositivi;
  2. in un contesto mobile la vicinanza con questi dispositivi non è ovvia.

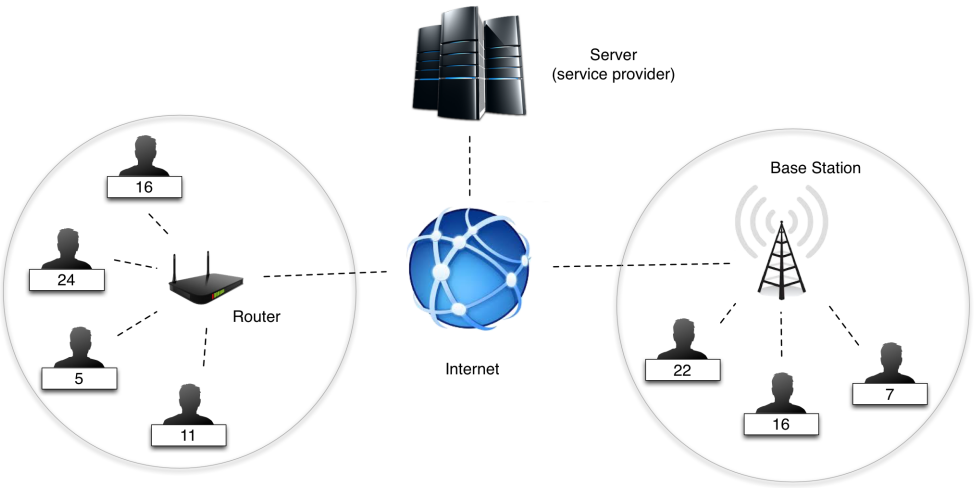


Figura 2.8: Architettura centralizzata per MSNs

Questo genere di architetture è un'estensione delle OSN web-based: utilizzando un'applicazione mobile o accedendo al sistema tramite un browser è possibile utilizzare tali sistemi.

L'architettura centralizzata è utilizzata con successo nelle wireless sensors networks (WSNs): i sensori sono degli strumenti che, attraverso l'utilizzo

dell'auto, degli smartphones, e di altre tecnologie, fanno parte della vita di tutti i giorni, e possono portare un gran numero di vantaggi integrando le WSNs con le MSNs. Utilizzando le informazioni provenienti dai sensori (la posizione, attraverso il GPS, la temperatura, tramite appositi sensori, etc) è possibile ottenere delle infomazioni più specifiche e personalizzate sugli utenti.

2.2.1.2 MSNs distribuite

L'aspetto più interessante delle MSNs distribuite è la totale mancanza di server centralizzati. Un nodo, in questo modo, è in grado di mantenere le proprie informazioni sociali e condividerle connettendosi con gli altri nodi. Sono gli utenti stessi a formare una vera e propria infrastruttura sulla quale basare le comunicazioni e lo scambio di informazioni, e si occupano anche della memorizzazione e dell'inoltro delle informazioni sociali (*store-and-forward*).

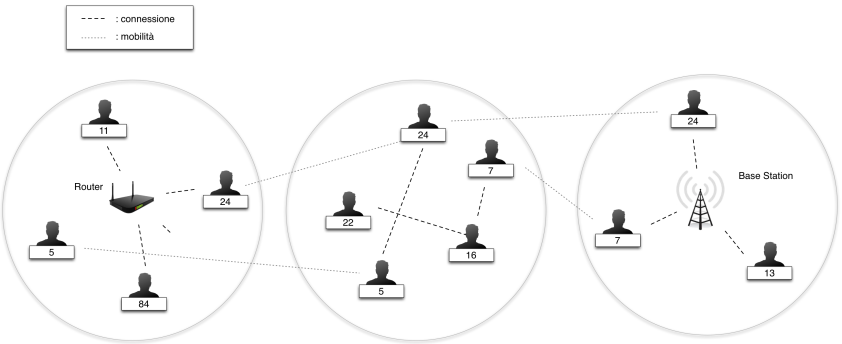


Figura 2.9: Architettura distribuita per MSNs

Per comunicare i vari dispositivi possono utilizzare stazioni radio base o access point, ma anche, vista la disponibilità di più interfacce wireless, come il Bluetooth e WiFi, connessioni ad hoc, senza la necessità di alcun tipo di infrastruttura (Figura 2.9). Possiamo quindi dire che si distinguono due tipologie di MSNs distribuite: quelle che utilizzano una qualche infrastruttura (gateway), e quelle che ne fanno a meno (connessioni ad-hoc), per effettuare le proprie comunicazioni. L'aspetto più interessante e competitivo di questa tipologia di reti è il secondo: fare in modo, utilizzando le interfacce messe a disposizione dai dispositivi mobili, di creare dei link senza utilizzare alcun tipo di access point.

In questo contesto, l'insieme dei dispositivi mobili può essere considerato come una rete ad hoc globale: un approccio di questo tipo è stato utilizzato

in [POL+09], con l'introduzione del middleware MobiClique. Questo framework è in grado di creare e matenere una rete ad hoc sociale, che consente lo scambio di informazioni utilizzando la mobilità degli utenti: tuttavia viene ignorato un possibile meccanismo di predizione dei contatti, e si utilizza esclusivamente flooding incontrollato. Per questo motivo MobiClique è quindi caratterizzato da inefficienza e da sovrautilizzo delle risorse. Nonostante questi problemi, MobiClique fornisce un chiaro esempio del funzionamento di una MSN con architettura completamente distribuita, fornendo un modello per quelli che saranno gli sviluppi futuri in questo ambito.

2.2.1.3 MSNs ibride

Le MSNs ibride (Figura 2.10) sono una combinazione delle architetture appena presentate, in cui si utilizza un paradigma centralizzato che contiene tutte le informazioni sociali sugli utenti, ed allo stesso tempo si effettuano anche comunicazioni ad-hoc tra i vari componenti della rete: questo indica che i dati sono memorizzati in un server centrale, ma allo stesso tempo sono condivisi in maniera distribuita tra i nodi della rete. Un esempio dell'uti-

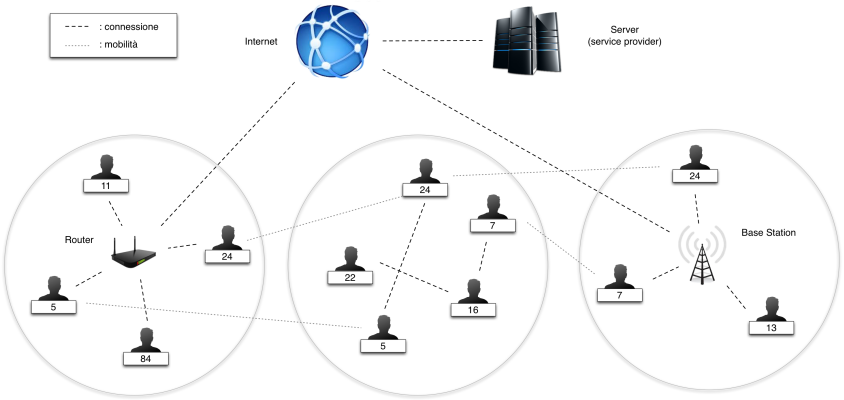


Figura 2.10: Architettura ibrida per MSNs

lizzo di questa infrastruttura ibrida si trova in [MPEP09]. In questo lavoro gli utenti che vogliono accedere ad una data informazione possono passare attraverso il server centralizzato via rete cellulare, oppure possono condividere informazioni tra loro con comunicazioni ad hoc. L'introduzione di un meccanismo di comunicazione ad hoc in una struttura centralizzata porta a dei notevoli miglioramenti, soprattutto per quando riguarda il bilanciamento del lavoro.

Un grande vantaggio di questo tipo di architetture è garantito dalla semplicità nell'estensione di una rete esistente con infrastruttura centralizzata o distribuita: in [HHK+12] è stato mostrato che, unendo questi due tipi di

infrastrutture si può arrivare ad ottenere una riduzione di traffico fino al 73.66%.

2.2.1.4 Componenti di una MSN

I nodi di una MSN (Figura 2.11) sono classificabili in tre tipologie:

- **Dispositivi Mobili:** sono l'interfaccia tra l'utente e questa tipologia di reti; attraverso il proprio dispositivo mobile un utente può accedere a contenuti disponibili sulla rete, o creare dei gruppi di utenti. Questi dispositivi possono essere dotati di una o più interfacce wireless per la comunicazione: ad esempio possono essere equipaggati con Bluetooth e WiFi, come generalmente accade nei dispositivi di ultima generazione.

Figura 2.11: Componenti di una MSN

- **Infrastruttura di rete:** infrastrutture fisse o mibili. Alcuni esempi sono le stazioni radio base e gli access point, dispositivi che danno l'opportunità di creare comunicazioni tra due (o più) dispositivi.
- **Content Providers:** in caso di architettura centralizzata questa componente è rappresentata dai server che contengono le informazioni sociali dei componenti delle MSNs, ed utilizzano le componenti dell'infrastruttura di rete per trasmetterle agli utenti. Anche i dispositivi possono essere visti come dei provider: infatti, nel caso distribuito, un dispositivo mobile è l'unico attore nella disseminazione di contenuti all'interno di una MSN.

2.2.2 Componente sociale nelle MSNs

L'aspetto chiave delle MSNs è la mobilità: i nodi di una rete sociale infatti si muovono in base ad obiettivi ed attività che derivano dalle proprie interazioni sociali [MSLC01, BP10]. Le interazioni sociali tra gli esseri umani sono spesso chiamate "legami sociali" (2.2.2.1); questi legami possono essere stretti, come quelli tra amici e familiari, o deboli, come quelli con sconosciuti. I legami sociali non sono un concetto molto semplice da descrivere e da modellare; in [Gra73] gli autori propongono una prima definizione basata sulla

combinazione di alcuni aspetti, come il tempo passato inseme, le emozioni coinvolte ed i vantaggi acquisiti dalle persone che compongono tale legame.

La tendenza a stare insieme ad altri individui che condividono gli stessi interessi è un'aspetto caratterizzante della natura umana, e questo determina la mobilità degli individui all'interno di una MSN. Di conseguenza anche la mobilità dei dispositivi che questi individui portano nelle loro tasche è direttamente condizionata da questo tipo di relazioni.

2.2.2.1 Legami sociali

I legami sociali tra le persone, nelle MSNs, determinano la topologia della rete. Con il termine "legame sociale" si vuole intendere una qualche relazione persistente o temporanea che coinvolge una coppia di individui. Esempi di legame sociale possono essere quello di amicizia, oppure la semplice conoscenza tra due persone. La distinzione tra queste due figure, ad esempio, può essere svolta analizzando alcuni aspetti della connessione tra i nodi:

- la frequenza di incontro: più spesso due persone si incontrano, maggiore tende ad essere il legame tra di loro;
- la confidenza, che rappresenta il tempo di durata di un contatto: più due contatti stanno insieme, maggiore tende ad essere il legame tra loro;
- il tempo di intercontatto, che descrive il tempo tra due contatti successivi: minore è il tempo di intercontatto tra due contatti, maggiore tende ad essere il legame tra loro;
- la regolarità, che rappresenta il fatto che un collegamento si ripeta nel tempo: un pattern ciclico indica un legame tra due contatti;
- l'omogeneità dal punto di vista sociale, ovvero gli interessi in comune tra due individui: due contatti che hanno molti interessi in comune è probabile che abbiano un legame di un qualche genere.

Utilizzeremo queste caratterisiche, in particolare la frequenza di contatto e la durata dei contatti, per classificare gli utenti incontrati. Inoltre, nel paragrafo 4.1.2, utilizzeremo delle metriche relative a quelle appena descritte per classificare le tracce di mobilità, in modo da comprendere quali potrebbero essere i risultati attesi.

Classificazione La frequenza e la durata degli incontri, per esempio, possono essere un indicatore del fatto che esista un legame sociale o no.

I rapporti tra gli individui di una MSNs possono essere generalmente suddivisi in quattro categorie principali, come possiamo vedere anche in Figura 2.12:

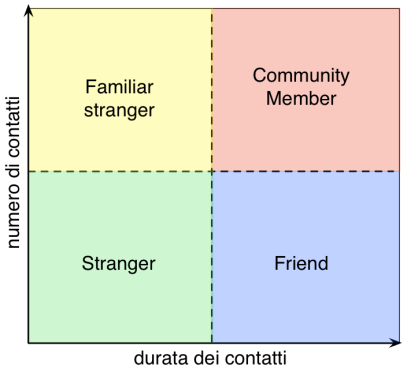


Figura 2.12: Categorizzazione del social neighborhood

- **Community Member**: descrive due utenti appartenenti alla stessa comunità. Un rapporto di questo tipo è generalmente caratterizzato da un gran numero di incontri di lunga durata. Generalmente questi incontri seguono patterns ciclici e periodici: un esempio di questo tipo contatti possono essere dei colleghi all'interno di un ufficio, dove gli individui si incontrano per tutta la giornata, tutti i giorni, e rimangono in contatto per tutte le ore di lavoro;

- **Familiar Stranger**: un rapporto di questo tipo è caratterizzato da un gran numero di incontri di breve durata. Anche in questo caso troviamo dei pattern ciclici: il gran numero di incontri tra due individui in questa categoria è dovuta proprio alla ciclicità del pattern. Un esempio di questo tipo di contatti può essere quello di due individui che tutte le mattine prendono la metro per andare a lavoro alla solita ora: questi due nodi potrebbero non avere interessi in comune e non conoscersi, però si incontrano tutte le mattine e stanno in contatto per un certo periodo.

- **Stranger**: è caratterizzato da un basso numero di incontri di breve durata. In questa categoria non ci sono pattern ciclici, anzi, in questo caso ci troviamo di fronte a pattern puramente casuali. Un esempio di questo tipo di contatti può essere quello tra due estranei che, casualmente, si trovano nello stesso negozio per fare acquisti.

- **Friend**: è caratterizzato da un basso numero di incontri di durata lunga. In questo caso nei pattern incontrati viene riscontrata grande periodicità: infatti, pensandoci, due amici si possono incontrare una o due volte a settimana, o al mese, a seconda dei propri impegni.

Di particolare interesse è quest'ultima categoria: infatti un rapporto di amicizia è influenzato anche da un aspetto che sarà molto importante per il

design di SIDEMAN$_{PQ}$, ovvero dagli interessi in comune tra gli utenti; due amici infatti, oltre alle caratteristiche temporali che legano due individui, sono in relazione anche per il numero di interessi in comune.

2.2.2.2 Community

La prima categoria mostrata nella classificazione appena vista è la più importante dal punto di vista delle interazioni sociali ed anche dal punto di vista delle possibili ottimizzazioni da svolgere su un protocollo su MSNs. Una comunità è un gruppo di individui i cui membri sono coinvolti in una qualche relazione sociale. Utilizzando il formalismo introdotto a inizio paragrafo, una comunità $\mathcal{C}^t$ nella quale si trovano i nodi $\{n_i, n_j, n_k\}$, sarà formata dall'unione di sottoinsiemi dei vicini dei nodi che la compongono, $\mathcal{C}^t \subseteq N_i^t \cup N_j^t \cup N_k^t$.

Le comunità offrono la possibilità di dividere le operazioni da eseguire per un certo protocollo, per esempio di routing, in due livelli: quello all'interno ed all'esterno della comunità (intra/inter community).

Per identificare le comunità all'interno delle MSNs esistono diversi algoritmi di community detection [BCP11, OF13] di diversa natura, distribuiti o centralizzati: analizzeremo questo aspetto nel parafrago 3.3.

2.3 Algoritmi di service discovery per reti ad-hoc e reti opportunistiche

I protocolli di service discovery sono studiati già da molti anni [KT04, VP08]. Molti di questi sono stati sviluppati per reti con infrastrutture statiche, dove i servizi sono offerti da nodi stazionari. Questo tipo di protocolli non soddisfa i requisiti e le caratteristiche tipici delle MSNs [BMD13].

Service discovery su reti mobili ad-hoc Per modellare delle soluzioni ideali per le MSNs è possibile sfruttare il lavoro svolto per i protocolli di service discovery nelle reti mobili ad-hoc. In [KKRO03] è stato implementato un algoritmo di service discovery basato sui Service Rings. Questa struttura raccoglie i nodi fisicamente vicini e che offrono servizi simili. L'accesso al service ring è garantito da un SAP (service access point) che memorizza le informazioni relative ai servizi offerti dai nodi della struttura. In questo modo è possibile evitare il flooding sull'intera rete per la ricerca di un servizio, ma basta inviare le richieste ai SAPs delle varie sottoreti. Questo approccio è utilizzato per ridurre l'overhead del protocollo; i SAPs svolgono un ruolo fondamentale nel protocollo, e devono essere riservate risorse per garantire a questi nodi di riuscire ad agire da ring manager.

Il protocollo GSD (Group-based Service Discovery) [CJYF06] introduce il concetto di gruppo di servizi, che stabilisce delle relazioni tra i servizi presenti nella rete: utilizza questo aspetto nella fase di selezione dell'advertisement data una specifica query (matching). GSD utilizza una schema di forwarding selettivo, dove una query viene propagata solo a nodi che offrono servizi che appartengono allo stesso gruppo di quello richiesto nella query, oppure che offrono tale servizio.

Gli autori di [HDVL03] hanno proposto un middleware progettato specificamente per la ricerca e la disseminazione di servizi all'interno di reti multihop ad-hoc. Si basa su un modello peer-to-peer ed utilizza una strategia di multicast basata su protocolli di routing per reti ad-hoc. Uno dei limiti di questo middleware è l'eccessivo costo dovuto all'overhead introdotto dal flooding utilizzato ogni volta che viene eseguito service discovery o service delivery.

In [NR07] è proposto un servizio di resource-discovery che utilizza aspetti sociali in DTNs, e che sfrutta il Community-based Mobility Model (CMM). Le risorse e gli utenti sono classificati in base ai propri interessi, ed una ricerca è eseguita inviando una query ai vicini con interessi simili: se questa strategia non ha successo si invia la query a tutti i nodi nel proprio range trasmissivo.

Service discovery su reti opportunistiche Negli ultimi anni nell'ambito della ricerca sono stati studiati algoritmi di service discovery per reti opportunistiche [PKO12]. Molti di questi lavori hanno tenuto in considerazione, nella fase di design degli algoritmi, la natura opportunistica e dinamica degli incontri tra i nodi. In [FPG12] sono stati eseguiti degli studi sull'impatto della limitazione delle risorse disponibili sulle performance in ambito opportunistico: in particolare è stato studiato l'impatto che la diversa topologia della rete ha sul possibile raggiungimento di servizzi ed accesso a risorse.

In [LSSM08] è stato proposto un modello basato sull'utilizzo di proxy per l'offerta di servizi in una rete opportunistica. L'idea principale è quella di considerare un servizo *proxable* se può essere fornito da più di un nodo nella rete: in questo modo un proxy si può comportare come un vero e proprio service provider, eseguendo la pubblicizzazione dei servizi, gestire le richieste di accesso ai servizi e le risposte.

In [LSM+11] è presentato un algoritmo opportunistico location-aware per discovery e accesso ai servizi. Gli autori hanno assunto che i nodi fossero a conoscenza della propria posizione geografica, e che fossero in grado di confrontare la posizione di due nodi. La posizione dei nodi è specificata per diversi scopi: *(i)* durante la fase di message forwarding, un messaggio è inoltrato solo ai nodi che sono vicini al destinatario *(ii)* durante la fase di service advertisement, un provider invia i propri advertisement esclusiva-

mente in una regione specifica vicino a lui *(iii)* durante la fase di invocazione del servizio, il messaggio di service request è inoltrato solo ai nodi vicini al provider.

L'ultimo lavoro che presentiamo è quello descritto in [MLSM12], dove gli autori hanno proposto il middleware Time-Aware Opportunistic (TAO) per reti intermittently connected, per accesso a servizi e loro richiesta. Utilizza euristiche per l'ottimizzazione della fase di service advertisement basata su un approccio cross-layer. Lo scenario considerato in TAO middleware riguarda un ambiente formato da nodi mobili, ma utilizza anche *info-stations* statiche per la diffuzione degli advertisement e per il providing dei servizi.

SIDEMAN e SIDEMAN$_{PQ}$ si distinguono dai protocolli presentati, in quanto *(i)* prendono in considerazione esclusivamente lo scenario delle MSNs, un ambiente distribuito e fortemente dinamico, *(ii)* considerano gli aspetti sociali dei nodi della rete, e *(iii)* costituiscono dei framework che si occupano esclusivamente del problema del service discovery.

Capitolo 3

Service Discovery in Mobile Social Networks

In questo capitolo presenteremo la progettazione di SIDEMAN$_{PQ}$, svolta durante la tesi: illustreremo prima l'algoritmo SIDEMAN, punto di partenza di questo lavoro di tesi, l'algoritmo SIDEMAN$_{PQ}$, ed infine gli algoritmi di community detection studiati ed analizzati per la fase di test dell'algoritmo.

3.1 SIDEMAN

Prima di descrivere l'algoritmo definiamo la notazione che utilizzeremo nel resto del capitolo.

La rete Una MSN è rappresentabile attraverso una serie di grafi temporali $G_t = (V, E_t)$, dove V indica l'insieme dei nodi, ed E_t indica gli archi attivi al tempo t. Per convenzione consideriamo $|V| = n$.

I nodi Ogni nodo n_i ha associato un insieme $\mathcal{I}_i = \{t_1, \ldots, t_s\}$, che descrive i propri interessi. L'insieme dei topics presenti in tutta la rete è definito con

$$\mathcal{I} = \bigcup_{j=1\ldots n} \mathcal{I}_j$$

Per convenzione consideriamo $|\mathcal{I}| = m$. Esempi di interessi sono lo sport, la cronaca, l'economia e la finanza, l'intrattenimento, etc.

Ogni nodo n_i ha una struttura chiamata *pending query* PQ_i, utilizzata per memorizzare le query per le quali non è stata ricevuta risposta, ed una service cache $\mathscr{A}_i$, nella quale sono memorizzati gli advertisement. Un nodo può inserire in cache advertisement *(i)* ricevuti da altri nodi durante incontri "opportunistici", oppure *(ii)* relativi a servizi erogati dal nodo stesso.

Ogni nodo mantiene aggiornata la propria contact-history T^i (tabella 3.1) per ogni nodo con cui ha avuto contatto nel tempo. Tale struttura memorizza diverse statistiche come la frequenza e la durata dei contatti tra il nodo n_i ed i nodi incontrati.

Nodo	**Last seen time** (sec)	**Contact duration** (sec)
n_j	10800	7200
n_k	500	180
n_w	54000	36000

Tabella 3.1: Esempio di contact-history del nodo n_i. La T^i ha il seguente significato: l'ultimo incontro tra n_i ed n_j è stato 3 ore fa (10800 secondi), e complessivamente sono stati insieme per 2 ore (7200 secondi); l'ultimo incontro tra n_i ed n_k è stato 5 minuti fa (500 secondi), e complessivamente sono stati insieme per 3 minuti (180 secondi); l'ultimo incontro tra n_i ed n_w è stato 15 ore fa (54000 secondi), e complessivamente sono stati insieme per 10 ore (36000 secondi).

I servizi Chiameremo $\mathcal{S} = \{s_1, \ldots, s_r\}$ l'insieme dei servizi presenti all'interno di una MSN. Ogni servizio è descritto da un advertisement. Tale advertisement contiene i seguenti campi: *(i)* un service identifier, identificativo univoco di un servizio all'interno della rete, *(ii)* un service provider, l'identificativo del nodo che offre tale servizio, ed *(iii)* un insieme di topics, che identificano gli interessi che corrispondono a quel servizio. [1]

Comunità	**Nodi della comunità**	**Interessi della comunità**
$\mathcal{C}_1$	$\{n_k, n_j\}$	$\{t_1, t_6, t_{17}\}$
$\mathcal{C}_2$	$\{n_j, n_w\}$	$\{t_1, t_2, t_8, t_{13}\}$
$\mathcal{C}_3$	$\{n_o, n_u\}$	$\{t_{15}, t_{18}\}$

Tabella 3.2: Tabella delle comunità di un nodo n_i

Le comunità Ogni nodo n_i tiene traccia delle comunità di cui ha fatto parte attraverso una struttura chiamata *community table*, $CT^i = \{\mathcal{C}_1, \ldots, \mathcal{C}_w\}$ (vedi tabella 3.2). La comunità $\mathcal{C}_i$ è rappresentata con l'insieme dei nodi che la compongono (paragrafo 3.3 mostra gli algoritmi di community detection).

In SIDEMAN, ogni volta che n_i incontra una nuova comunità $\mathcal{C}$ la inserisce in CT^i solo se non è presente un'altra comunità $\mathcal{C}' \in CT^i$ *simile* a $\mathcal{C}$. Per calcolare la similarità tra due comunità è stato utilizzato l'indice di Jaccard

[1] Non è scopo di questo lavoro di tesi investigare nel dettaglio la descrizione dei service advertisement.

[SM12]: in base a questo indice, due comunità $\mathcal{C}$ e $\mathcal{C}'$ sono considerate simili se

$$J(\mathcal{C}, \mathcal{C}') = \frac{|\mathcal{C} \cap \mathcal{C}'|}{|\mathcal{C} \cup \mathcal{C}'|} > \tau$$

per una data soglia di similarità τ.

Ogni volta che una nuova comunità C_n è inserita in CT, è memorizzato anche l'insieme degli interessi $\mathcal{I}_n$ della comunità; in questo modo, quando un nodo entra a far parte di una comunità già presente in CT, conosce anche gli interessi dei nodi che ne fanno parte.

3.1.1 Panoramica

SIDEMAN è un algoritmo utilizzato da ogni nodo $n \in V$ per la diffusione e la ricerca di servizi in una MSN. SIDEMAN prevede che i nodi *(i)* in maniera reattiva inviino query per servizi per i quali non hanno advertisement nella propria cache, *(ii)* in maniera proattiva scambino query $q \in PQ_i$ che potrebbero ricevere risposta dai nodi vicini, ed *(iii)* in maniera proattiva scambino advertisement $adv \in \mathscr{A}_i$ di interesse per i nodi in contatto.

Mentre si sposta ed incontra altri nodi, un nodo n_i esegue periodicamente un algoritmo per il calcolo della comunità alla quale appartiene. Una volta determinata, n_i controlla se ne è presente una *simile* in CT^i: se non la riconosce, allora la inserisce all'interno della sua tabella delle comunità CT^i. Le comunità, negli algoritmi SIDEMAN e SIDEMAN$_{PQ}$, sono molto importanti, e sono utilizzate soprattutto nella fase di inoltro dei messaggi.

La fase reattiva e proattiva di SIDEMAN sono mostrate di seguito.

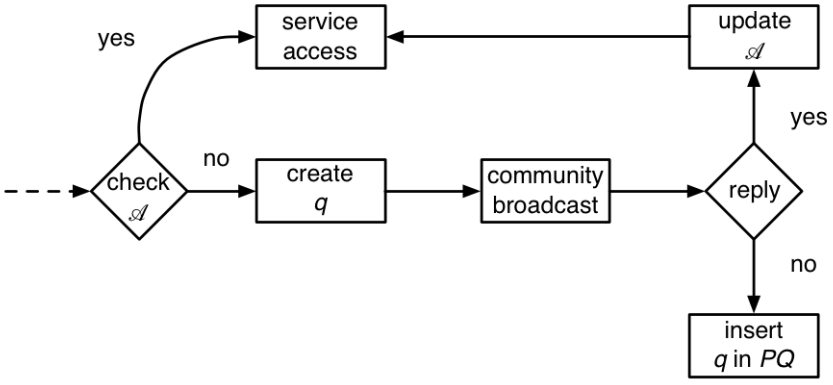


Figura 3.1: Fase reattiva in SIDEMAN

Fase reattiva: (Figura 3.1) se un nodo n_i vuole accedere ad un servizio, controlla se ha un advertisement corrispondente eseguendo una query q alla propria service cache $\mathscr{A}_i$. Se ha una corrispondenza, allora accede al servizio,

altrimenti inoltra la query ai nodi della sua comunità che condividono almeno un interesse con quelli della query (controlled flooding). Se qualcuno tra i nodi che hanno ricevuto il messaggio ha una risposta alla query, allora la invia ad n_i, che la salverà all'interno della sua service cache $\mathscr{A}_i$, altrimenti la query è inserita nella struttura dati PQ_i in attesa di futuri incontri.

Fase proattiva: (Figura 3.2) ogni volta che n_i riconosce una comunità, tale nodo scambia degli advertisement in base agli interessi della comunità stessa, memorizzati all'interno di CT^i. I nuovi advertisement ricevuti sono

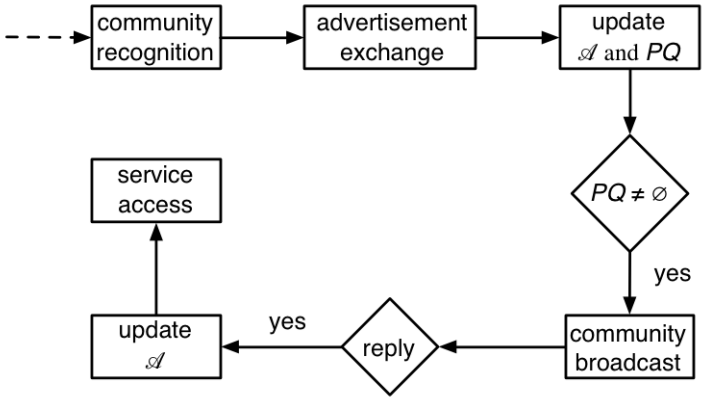


Figura 3.2: Fase proattiva in SIDEMAN

memorizzati nella service cache $\mathscr{A}_i$. A questo punto n_i controlla se tra i nuovi advertisement qualcuno può rispondere alle query $q \in PQ_i$: le query che ricevono risposta sono cancellate da PQ_i, e le rimanenti sono inoltrate ai membri della comunità. Quando n_i ha a disposizione l'advertisement per un servizio richiesto vi può accedere.

3.1.2 Algoritmo

Prima di eseguire sia la fase proattiva, sia quella reattiva, un nodo n_i esegue l'algoritmo A di community detection distribuito. Questo algoritmo utilizza come parametri per il calcolo: T, la contact-history di un nodo, e N, l'insieme dei vicini del nodo al tempo in cui si sta eseguendo l'algoritmo. La comunità $\mathcal{C}$ calcolata dall'algoritmo 1 è quella dove si trova il nodo n_i: questo procedimento è eseguito ad intervalli periodici. La comunità $\mathcal{C}$ è *riconosciuta* se troviamo una comunità $\mathcal{C}'$ simile a $\mathcal{C}$ all'interno della sua community table CT^i. Come già detto, la similarità tra due comunità è calcolata attraverso l'indice di Jaccard, confrontato con una soglia stabilita τ. Se è presente in CT^i una comunità $\mathcal{C}'$ simile a $\mathcal{C}$, allora vengono assegnati a $\mathcal{C}$ gli interessi di $\mathcal{C}'$ (riga 3), altrimenti gli interessi dei $\mathcal{C}$ sono collezionati attraverso l'invio

Algoritmo 1 RecognizeCommunity(T, N, CT, τ)

1: $\mathcal{C} = A(T, N)$
2: **if** $\exists\ \mathcal{C}' \in CT \mid J(\mathcal{C}, \mathcal{C}') \geq \tau$ **then**
3: $\quad \mathcal{I}_{\mathcal{C}} =$ interests of $\mathcal{C}'$ from CT
4: **else**
5: $\quad \mathcal{I}_{\mathcal{C}} = getInterests(\mathcal{C})$
6: $\quad CT = CT \cup \mathcal{C}$
7: **end if**
8: return $\mathcal{C}$

di una richiesta ad ogni nodo della comunità. Questi invii sono eseguiti all'interno della funzione $getInterests(\cdot)$ (riga 5). Una volta ottenuta questa informazione la comunità $\mathcal{C}$ può essere memorizzata all'interno di CT^i (riga 6). Ogni nodo, in questo modo, memorizza tutte le comunità visitate e gli interessi associati.

Il modo con il quale è calcolato l'insieme dei topics della comunità $\mathcal{C}$ può richiedere un gran numero di comunicazioni, soprattutto quando un nodo ha molti vicini: questo può portare ad un degrado delle performance, a tempi di attesa maggiori e maggiore consumo energetico. Tuttavia i test eseguiti su SIDEMAN hanno mostrato che più del 50% delle volte in cui è eseguito l'algoritmo 1, la funzione $getInterests(\cdot)$ non è richiamata. Questo significa che molto spesso le comunità sono riconosciute.

Dopo aver riconosciuto la comunità nella quale si trova, n_i può ricercare l'advertisement per il servizio di cui ha bisogno come mostrato nell'algoritmo 2 (fase reattiva).

Algoritmo 2 DiscoverService(q)

1: $\mathcal{C} =$ RecognizeCommunity(T^i, N^i, CT^i, τ)
2: **if** $\exists adv_j \in \mathscr{A}_i$ s.t. $adv_j =$ reply(q) **then**
3: $\quad$ access $service(adv_j)$
4: **else**
5: $\quad V_q = \{n_j \in \mathcal{C} \cap N^i \mid \exists_1\ t_k \in \mathcal{I}_q$ s.t. $t_k \in I_j\}$
6: $\quad forwardQuery(q, V_q)$
7: **end if**

Nella versione base di SIDEMAN non è stato ideato alcun metodo per le fasi di selezione ed accesso: dopo la ricerca nella service cache $\mathscr{A}_i$ di advertisement corrispondendi alla query (riga 2), se n_i trova in cache una risposta, allora utilizza tale advertisement in modo da raggiungere il servizio richiesto (riga 3), altrimenti calcola l'insieme $V_q \subseteq \mathcal{C}$ che contiene i nodi della sua comunità in contatto con lui che potrebbero potenzialmente rispondere

alla query (riga 5). Infine la query viene inviata in broadcast ai nodi in V_q (riga 6).

La fase di costruzione dell'insieme V_q può essere costosa (in termini di energia e tempo spesi). Per questo motivo gli insiemi degli interessi sono stati implementati utilizzando una struttura dati succinta, i Bloom Filters (BFs) [TRL12], in modo da ottimizzare questa fase: il controllo della presenza di un elemento nei BFs è eseguito in tempo costante.

Algoritmo 3 ForwardAdvertisementsAndQueries

```
1:  C = RecognizeCommunity(T^i, N^i, CT^i, τ)
2:  for all t ∈ I_C do
3:      V_t = {n_j ∈ C ∩ N^i | t ∈ I_j}
4:      if V_t = ∅ then
5:          I_C = I_C \ {t}
6:          break
7:      end if
8:      ADV_t = {adv_j ∈ A_i | t ∈ adv_j.getTopics()}
9:      forwardAdvertisements(ADV_t, V_t)
10:     for all q ∈ PQ_i do
11:         V̂_t = {n_j ∈ C ∩ N^i | t ∈ I_j}
12:         forwardQuery(q, V̂_t)
13:     end for
14: end for
```

Nella fase proattiva invece, SIDEMAN scambia gli advertisement e le query presenti nella struttura PQ: questo procedimento è descritto dall'algoritmo 3.

Una volta che n_i ha riconosciuto la propria comunità, per ogni topic $t \in \mathcal{I}_\mathcal{C}$, n_i:

1. calcola l'insieme $V_t \subseteq \mathcal{C} \cap N^i$ dei nodi in $\mathcal{C}$ e in contatto con n_i che sono interessati al topic t (riga 3). Se V_t è vuoto, n_i rimuove t da $\mathcal{I}_\mathcal{C}$ (riga 5): in questo modo è possibile gestire la dinamicità della struttura e degli interessi delle comunità. Se invece V_t contiene dei nodi, allora n_i calcola l'insieme $ADV_t \subseteq \mathscr{A}_i$ degli advertisement che hanno tra i loro interessi t (riga 7).

2. inoltra gli advertisement ADV_t a tutti i nodi in V_t (riga 8).

3. seleziona le query in PQ_i che contengono il topic t e le inoltra all'insieme $\hat{V}_t$ (riga 12). Questo insieme è formato dai nodi in $\mathcal{C}$ e in contatto con n_i che sono interessati al topic t (riga 11).

Come prima, gli insiemi V_t e $\hat{V}_t$ sono implementati in maniera efficiente utilizzando dei BFs. Sia nel caso rettivo (algoritmo 2) sia in quello proattivo

(algoritmo 3), dopo l'inoltro delle query e degli advertisement alcuni vicini di n_i potrebbero trasmettere alcune risposte (algoritmo 4).

Algoritmo 4 OnMessageReception(m)

1: **if** m is a query **then**
2: Extract from $\{adv_1^i, \ldots, adv_k^i\} \subseteq \mathscr{A}_i$ matching query m
3: Unicast($\{adv_1^i, \ldots, adv_k^i\}$, sender($m$))
4: **else**
5: Update($\mathscr{A}_i$, m)
6: **end if**

Quando riceve un messaggio m, il nodo n_i controlla se si tratta di una query o di un advertisement. Se m è una query allora controlla in $\mathscr{A}_i$ quali advertisements possono servire a rispondere alla query, e li invia al mittente della query (righe 2-3). Se m è un advertisement, aggiorna la sua service cache. La funzione Update($\mathscr{A}_i$, m) controlla se l'advertisement ricevuto è già in cache: se è così, aggiorna l'advertisement, altrimenti lo inserisce in $\mathscr{A}_i$.

3.1.3 Analisi di SIDEMAN

L'algoritmo SIDEMAN offre una possibile soluzione al problema del service discovery su MSNs. SIDEMAN adotta un approccio reattivo per permettere ai service client di ricevere gli advertisement di cui fanno richiesta, ed uno proattivo per permettere ai service client di diffondere le proprie query che non hanno ricevuto risposta, ed ai service provider di pubblicizzare i propri servizi.

Al momento:

- la struttura PQ è utilizzata in SIDEMAN per permettere ai service client di conoscere quali tra le *proprie query* non hanno ancora ricevuto risposta. In SIDEMAN$_{PQ}$ questa struttura sarà sfruttata in maniera differente, cercando di migliorare le prestazioni, soprattutto il tempo di risposta delle query.
- in SIDEMAN non sono stati specificati particolari meccanismi per le fasi di selezione e di accesso.

Oltre all'ottimizzazione dell'algoritmo, in questa tesi ci occuperemo anche di dotare SIDEMAN$_{PQ}$ di questi due meccanismi, in modo da completare l'algoritmo di service discovery.

3.2 SIDEMAN$_{PQ}$

In questo paragrafo descriveremo SIDEMAN$_{PQ}$, un algoritmo di service discovery e service dissemination per MSNs. In SIDEMAN$_{PQ}$ sono stati

affrontati alcuni aspetti non studiati, o considerati in maniera minore, in SIDEMAN:

1. sono state formalizzate in mainera più precisa le strutture presenti nell'algoritmo;
2. sono stati chiariti gli obiettivi delle diverse attività svolte nell'algoritmo;
3. sono state create delle modalità di invio basate sulla storia temporale degli incontri tra nodi (*temporal distance*), e sui loro interessi (*social centrality*);
4. sono state ristrutturate la fase proattiva/reattiva, e le fasi di gestione dei messaggi;
5. è stato implementato un meccanismo per le fasi di selezione ed accesso.

3.2.1 Strutture utilizzate in SIDEMAN$_{PQ}$

Come già detto durante la descrizione di SIDEMAN, l'algoritmo utilizza due strutture dati per memorizzare query ed advertisement: *pending queries* (PQ) e *service cache* ($\mathscr{A}$). Query ed advertisement sono a loro volta delle strutture dati.

3.2.1.1 Query e Pending Query

Una query contiene le informazioni mostrate anche in Figura 3.3:

- **Query ID (id_q):** identificativo univoco della query all'interno della rete.
- **Interests set ($\mathcal{I}_q$):** insieme dei topic che riguardano la query, utilizzato nella fase di controllo della presenza di risposte (ricerca di advertisement con interessi corrispondenti).
- **Requester node (n_q):** mittente della query, utilizzato dal nodo che risponderà alla query per inoltrare la risposta. Questo campo è necessario, dato che le query spesso sono inoltrate in maniera indiretta.
- **Creation Time ($time_q$):** timestamp eseguito alla creazione della query.
- **Time To Leave (TTL_q):** TTL, espresso in numero di hop, di una query. Dopo TTL inoltri tra nodi la query è ritenuta obsoleta e non è più inoltrata.

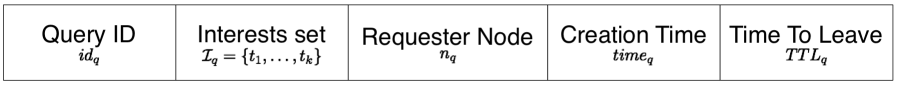


Figura 3.3: Struttura dati di una *query*

In SIDEMAN le query che non ricevono risposte sono inserite nella struttura dati PQ. Nel caso di SIDEMAN$_{PQ}$ questa struttura ha un funzionamento più articolato: infatti un nodo n_i, quando riceve una query un nodo n_j a cui non può ripondere, la inserisce nella propria struttura PQ_i. In questo modo il protocollo cerca di sfruttare il fatto che n_i potrebbe incontrare un'altro nodo n_k in grado di fornire una risposta alla query q, e successivamente inoltrare la risposta a n_j (*"friend-of-friends"*, paragrafo 1.1).

Quindi nella struttura PQ_i di n_i sono presenti, oltre alle query generate dal nodo n_i, anche quelle provenienti da altri nodi, ed alle quali non è stato possibile rispondere.

3.2.1.2 Advertisements e Service Cache

Un advertisement (Figura 3.4) è una struttura dati formata da:

- **Advertisement ID (id_{adv}):** identificativo univoco dell'advertisement all'interno della rete.
- **Interests set ($\mathcal{I}_{adv}$):** insieme dei topic che riguardano il servizio, utilizzato in fase di risposta ad una query. Se la query ricevuta q ha interessi ($\mathcal{I}_q$) che corrispondono con quelli di un advertisement, allora si utilizza tale advertisement come risposta.
- **Service ID (s):** identificativo univoco del servizio all'interno della rete.
- **Service provider ($provider_s$):** identificativo univoco del provider del servizio.
- **Recipient (n_{adv}):** destinatario dell'advertisement; può corrispondere all'identificativo di un nodo, ma può essere anche vuoto. Vedremo più avanti il significato di questo campo.
- **Validity (t):** tempo di creazione dell'advertisement.

Figura 3.4: Struttura dati di un *advertisement*

Il campo *validity* è stato inserito per distinguere due advrtisement per lo stesso servizio: più grande è il valore di questo campo, più aggiornato è l'advertisement.

È stato inoltre introdotto un campo *recipient*, utilizzato per distinguere gli advertisement in due classi: quelli con il campo *recipient* non vuoto saranno inviati al destinatario (come risposta ad una query, per esempio), gli altri saranno inoltrati nella fase di service dissemination.

3.2.2 Obiettivi in SIDEMAN$_{PQ}$

In SIDEMAN$_{PQ}$ sono stati analizzati gli obiettivi principali dei vari step del service discovery (Figura 2.2, paragrafo 2.1.2): query, advertisement, service request, e serivce response.

Gli obiettivi associati all'inoltro di questi messaggi sono diversi per ogni tipologia: ogni situazione è stata analizzata per cercare di migliorare gli aspetti identificati.

Query L'obiettivo della fase di query è quello di inviare le query ai nodi con la maggior probabilità di risposta. Un nodo è in grado di rispondere ad una query se e solo se ha uno o più advertisement all'interno della sua service cache con delle corrispondenze con gli interessi della query.

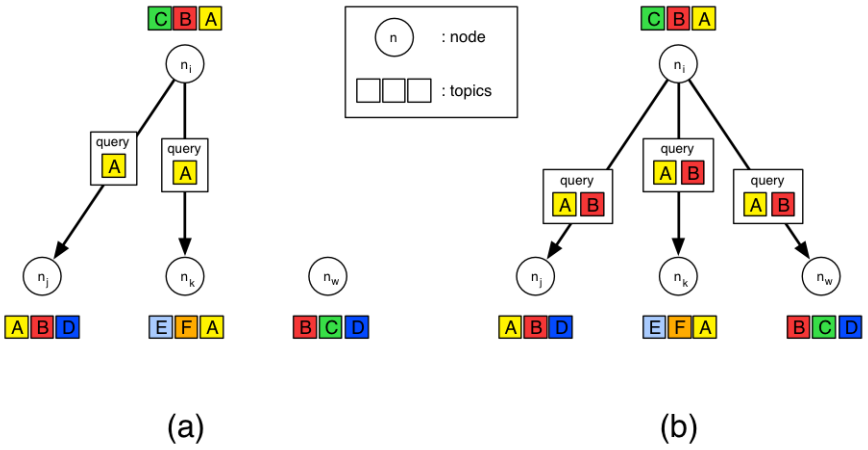


Figura 3.5: Esempio di inoltro di query basato sugli interessi

Per massimizzare la probabilità di risposta, evitando il flooding incontrollato, il nodo n_i deve inviare la query q ad un insieme di nodi che hanno interessi in comune con quelli in $\mathcal{I}_q$. Utiliziamo gli esempi in Figura 3.5 per chiarire questo aspetto. Nel primo esempio (Figura 3.5a), la strategia utilizzata per il forwarding della query utilizza una corrispondenza 1:1 tra gli interessi: n_i inoltra la propria query ad n_j ed n_k, dato che questi due nodi

hanno tra i loro topic di interesse quelli di q, in particolare n_j ed n_k condividono il topic A con q. In Figura 3.5b è mostrato un caso più complesso: n_i decide di inoltrare la propria query a tutti i vicini, dato che hanno all'interno del loro insieme di interessi almeno uno dei topic della query.

Questa strategia di inoltro delle query può essere generalizzata in questo modo: scegliamo una soglia τ che identifica un valore minimo di similarità al di sotto del quale la query q non è inviata. Consideriamo, ad esempio, di utilizzare l'indice di Jaccard (quindi $0 \leq \tau \leq 1$): il nodo n_i verifica la similarità tra gli interessi di q e quelli dei nodi n_j, n_k ed n_w

$$J(\mathcal{I}_q, \mathcal{I}_j) > \tau \quad \wedge \quad J(\mathcal{I}_q, \mathcal{I}_k) < \tau \quad \wedge \quad J(\mathcal{I}_q, \mathcal{I}_w) > \tau$$

In questo caso n_i invierà la query q a n_j e n_w, scartando n_k, che non è ritenuto idoneo.

L'esempio in Figura 3.5b potrebbe utilizzare questa strategia con $\tau \leq \frac{1}{4}$; infatti dato che :

$$J(\mathcal{I}_q, \mathcal{I}_x) = \frac{|\mathcal{I}_q \cap \mathcal{I}_x|}{|\mathcal{I}_q \cup \mathcal{I}_x|}$$

avremo che:

$$J(\mathcal{I}_q, \mathcal{I}_j) = \frac{|\{A, B\}|}{|\{A, B, D\}|} = \frac{2}{3}$$

$$J(\mathcal{I}_q, \mathcal{I}_k) = \frac{|\{A\}|}{|\{A, B, E, F\}|} = \frac{1}{4}$$

$$J(\mathcal{I}_q, \mathcal{I}_w) = \frac{|\{B\}|}{|\{A, B, C, D\}|} = \frac{1}{4}$$

Advertisement L'inoltro degli advertisement ha due obiettivi principali, a seconda del tipo di advertisement. Un advertisement può essere inoltrato ad un nodo per due motivi:

1. come risposta ad una query (in seguito ad una fase reattiva);
2. per diffondere informazioni sul servizio descritto dall'advertisement all'interno della rete (fase proattiva).

Nel primo caso, l'obiettivo è quello di recapitare il messaggio al destinatario (il mittente della query) il prima possibile. Nel secondo caso vogliamo recapitare i messaggi a nodi che sono effettivamente interessati agli advertisement, ovvero tali che ci sia una corrispondenza tra l'insieme dei topics dell'advertisement e quello dei nodi.

Service request/response L'inoltro di questo tipo di richieste ha un obiettivo comune, ovvero quello di far giungere il prima possibile il messaggio a destinazione:

- nel caso di un messaggio di service request, il nodo che richiede l'accesso al servizio deve essere in grado di recapitare il prima possibile al service provider tale messaggio;
- nel caso di un messaggio di service response, il nodo che fornisce il servizio deve essere in grado di fornire una risposta il prima possibile al service client.

3.2.3 Strategie di forwarding

Stabiliti gli obiettivi principali delle fasi di service discovery è necessario ora definire le strategie di forwarding di SIDEMAN$_{PQ}$. Le novità principali introdotte rispetto a SIDEMAN sono: la *Social Centriality* (SC) e la *Temporal Distance* (d). In questo paragrafo mostreremo queste due metriche per la scelta dei migliori candidati nel forwarding dei messaggi (query, advertisement, service request/response).

3.2.3.1 Social Centrality

La *Social Centriality* è una metrica che tenta di fondere metriche temporali e sociali per scegliere il nodo a cui inoltrare un messaggio: *(i)* le metriche temporali sono utilizzate per la selezione di contatti che un nodo può incontrare con maggior frequenza, per maggior durata, o secondo altri parametri temporali, mentre *(ii)* le metriche sociali sono utilizzate per la selezione di contatti più simili, ad esempio, che condividono un maggior numero di interessi, o che frequentano gli stessi luoghi, etc. La *Social Centriality* cerca di unire questi due aspetti assieme.

Infatti, i protocolli di service discovery che si basano esclusivamente o su metriche temporali, o su metriche sociali, possono avere dei problemi:

- **Metriche temporali:** nodi che si incontrano spesso possono non avere però molti interessi in comune. Utilizzando esclusivamente le metriche temporali, due nodi simili da un punto di vista sociale, che però non si incontrano spesso, possono non scambiarsi messaggi.
- **Metriche sociali:** nodi socialmente molto simili potrebbero non incontrarsi mai, e quindi non scambiarsi messaggi.

La *Social Centrality*, quindi, misura la capacità di un nodo, durante un contatto, di propagare in maniera efficiente un messaggio all'interno della rete. Nella *Social Centrality* utiliziamo il concetto di *"friend-of-friends"* espresso nel paragrafo 1.1; infatti la SC considera:

- la *similarità* tra la query ed i topics di interesse del nodo candidato;
- la *similarità* tra la query ed i topics di interesse degli "amici" del nodo candidato (i componenti della sua local community);
- il tempo di medio di ricontatto con il candidato.

A questo punto possiamo definire la *Social Centrality*.

Dati:

- $s_{q,j}$ la similarità tra la query q ed il nodo n_j (indice di Jaccard)
- $\mathcal{C}_j$ la local community del nodo n_j (t.c. $n_j \in \mathcal{C}_j$)
- t_r il tempo medio di ricontatto tra i nodi n_i e n_j, tale che:

$$t_r(i,j) = \begin{cases} 0 & e^t_{i,j} = 1 \\ \geq 0 & \text{altrimenti} \end{cases} \tag{3.1}$$

allora possiamo definire la *Social Centrality* come:

$$SC(q) = \sum_{\substack{w \in \mathcal{C}_j \\ r_r(i,w) > t_r(j,w)}} \frac{s_{q,w}}{1 + t_r(i,w)} \tag{3.2}$$

Definendo in questo modo SC, consideriamo sia gli aspetti temporali, che quelli sociali, permettendo agli uni di influenzare gli altri.

La *Social Centrality* è utilizzata in fase di forwarding delle query, quindi in modalità reattiva ed in modalità proattiva. Questa strategia è stata adottata per tentare di seguire gli obiettivi menzionati nel paragrafo 3.2.2, ovvero, nel caso del forwarding delle query, minimizzare il tempo di risposta ad una query.

3.2.3.2 Temporal Distance

La *temporal distance* permette di misurare quanto "velocemente" un nodo incontrerà un altro nodo, utilizzando il tempo medio di ricontatto tra due nodi (formula 3.1), ed il tempo in cui l'ultimo contatto è avvenuto. La time distance tra il nodo n_x ed il nodo n_y al tempo t è calcolata come

$$d_{x,y} = t^x_r(y) - (t - t^x_l(y))$$

Dove:

- $t^x_r(y)$: il *tempo medio di ricontatto* tra il nodo n_x ed n_y

- $t_l^x(y)$: il *tempo dell'ultimo incontro* tra il nodo n_x ed n_y

I valori $t_r^x(y)$ e $t_l^x(y)$ sono memorizzati all'interno del record della contact-history $T^x(y)$. Questa metrica misura quanto dovrà attendere il nodo n_x prima di incontrare nuovamente il nodo n_y. Per poter calcolare la temporal distance, i nodi devono memorizzare i valori del tempo medio di ricontatto e del tempo dell'ultimo incontro per ogni nodo incontrato: tali valori possono essere memorizzati nella contact-history T^i, modificando opportunamente la struttura dati precedente (tabella 3.1).

L'algoritmo 5 descrive come aggiornare la *contact-history*. La procedura funziona come segue:

- n_i rileva i nodi con cui è in contatto (neighborhood N^i);
- se n_i ha un record nella propria tabella per il nodo $n_j \in N^i$, allora aggiorna le caratteristiche temporali di n_j (algoritmo 5, righe 4-8), altrimenti aggiunge un nuovo record per tale nodo, inizializzandone i campi (algoritmo 5, righe 10-12);
- l'operazione di aggiornamento del record prevede la modifica del solo campo $t_l^i(j)$, nel caso in cui il nodo n_j sia già in contatto con n_i anche nell'unità temporale precedente, altrimenti è modificato anche il campo $t_a^i(j)$ (algoritmo 5, righe 4-7).

Algoritmo 5 Aggiornamento contact-history di n_i

```
1:  t = getTime()
2:  for all n_j ∈ N^i do
3:      if ∃ entry for n_j : T^i(j) then
4:          if t_l^i(j) ≠ (t − 1) then
5:              newMeanValue = t − t_l^i(j)
6:              updateMeanValue(d_{i,j}, newMeanValue)
7:          end if
8:          t_l^i(j) = t
9:      else
10:         create new record T^i(j)
11:         t_l^i(j) = t
12:         updateMeanValue(d_{i,j}, newMeanValue)
13:     end if
14: end for
```

Utiliziamo l'esempio in Figura 3.6: nella figura è mostrata l'evoluzione, durante il tempo, dei contatti tra quattro nodi, in cui gli incontri tra i nodi sono rappresentati dalle frecce. In figura sono riportate anche le contact-history dei nodi B e C. Il nodo A deve scegliere, al tempo $t = 80$, il miglior

candidato per inoltrare un messaggio al nodo D. Inizialmente A richiede a B e C i valori presenti nelle proprie *contact-history* riguardanti il nodo D, e calcola la temporal distance per B

$$d_{B,D} = 26 - (80 - 79) = 25$$

e per C

$$d_{C,D} = 20.5 - (80 - 69) = 9.5$$

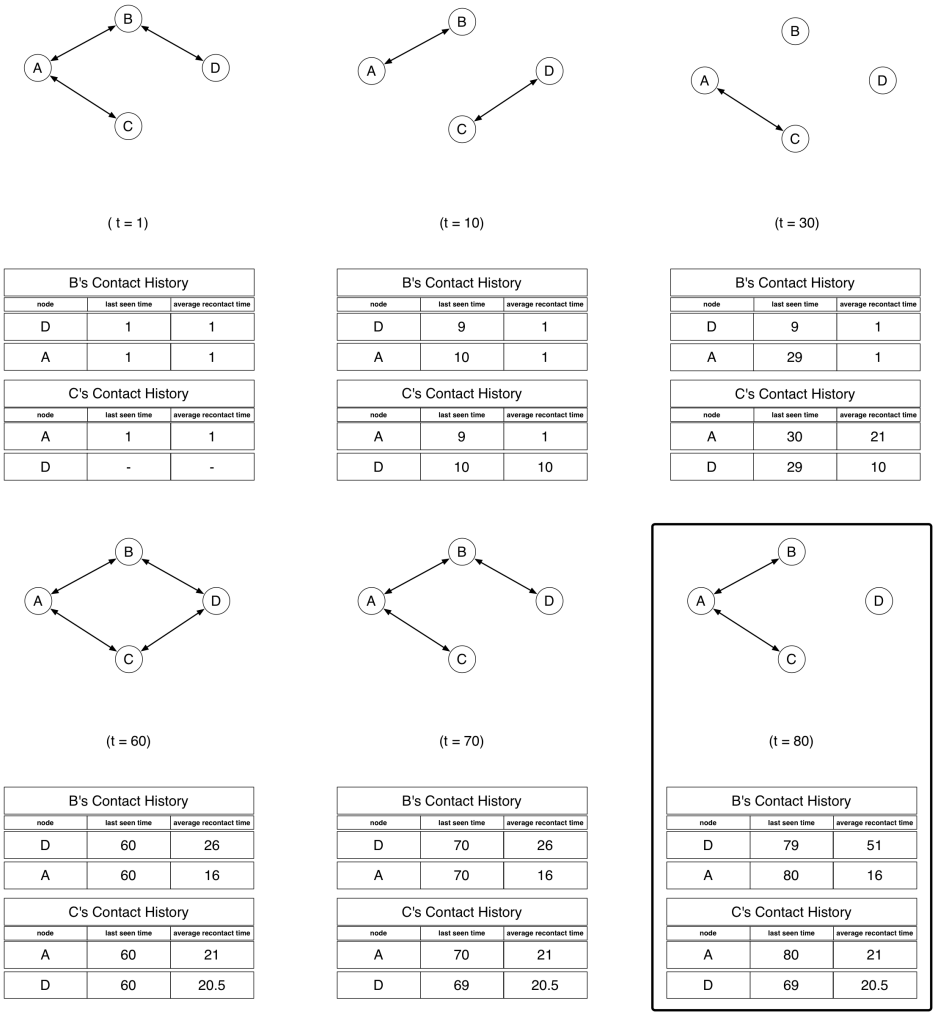


Figura 3.6: Esempio calcolo *temporal distance*

Utilizzando questa strategia il nodo A sceglierà C come nodo a cui inoltrare il messaggio. I nodi B e C sono stati in contatto con D la stessa quantità di tempo, tuttavia il nodo C ha incontrato D con una maggiore frequenza rispetto a B: in Figura 3.6 possiamo notare che il primo contatto tra

B e D è terminato al tempo 9, ed il secondo contatto è iniziato al tempo 60, con un tempo di intercontatto pari a $60 - 9 = 51$, mentre il primo contatto tra C e D è terminato al tempo 29, ed il secondo è iniziato al tempo 60, con un tempo di intercontatto pari a $60 - 29 = 31$.

La temporal distance è utilizzata in molte circostanze in SIDEMAN$_{PQ}$: nella fase proattiva dell'invio degli advertisement, nella fase di accesso al servizio, ed in tutte le circostanze in cui è necessario inviare un messaggio ad un destinatario non in contatto. In generale questa tecnica è applicata per tutti i tipi di messaggi, ad esclusione delle query. Tutte le volte che un nodo deve inviare un messaggio con uno specifico destinatario utilizza questa tecnica.

3.2.4 Componente reattiva

La fase reattiva in SIDEMAN$_{PQ}$ è gestita come mostrato in Figura 3.7: è utilizzata la stessa procedura di riconoscimento della comunità riportata anche nell'algoritmo 1 (pag. 30). In questa fase un nodo n_i crea una query

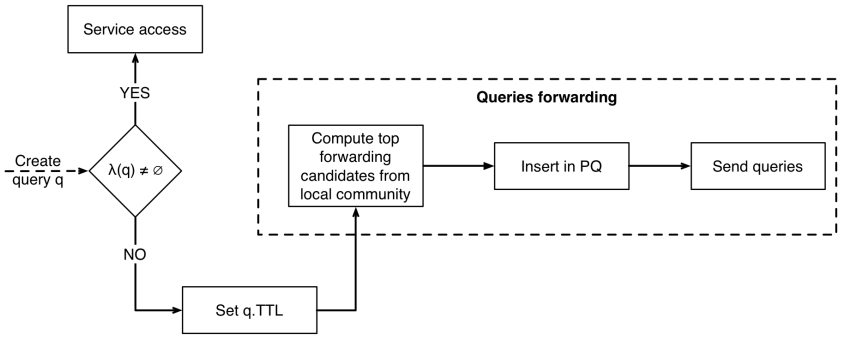


Figura 3.7: Fase reattiva in SIDEMAN$_{PQ}$

q, e controlla se esistono degli advertisement per rispondere a q, eseguendo la funzione di selezione λ (algoritmo 6, riga 2). La funzione λ, data una query, seleziona il "miglior" advertisement che massimizza una certa funzione obiettivo. In SIDEMAN$_{PQ}$ la funzione di selezione è stata implementata in modo da ricercare all'interno della service cache un advertisement che massimizzi la similarità tra il proprio insieme di interessi e quello della query:

$$\lambda(q) = \max_{\forall adv_k \in \mathscr{A}_i} Jaccard(\mathcal{I}_{adv_k}, \mathcal{I}_q)$$

Se la funzione di selezione restituisce un advertisement, allora n_i può accedere al servizio inviando un messaggio di service request (algoritmo 6, righe 3-5).

Algoritmo 6 DiscoverService(q)

1: $C = \text{RecognizeCommunity}(T^i, N^i, CT^i, \tau)$
2: **if** $\lambda(q) = adv_j \in \mathscr{A}_i$ **then**
3: m = $composeServiceRequest(adv_j)$
4: dst = $r(m)$
5: Send service request m to dst
6: **else**
7: $V_q = computeSocialCentrality(q, N^i)$
8: $forwardQuery(q, V_q)$
9: **end if**

La selezione di questo nodo è svolta utilizzando la funzione di accesso r. L'obiettivo della funzione r è quello di minimizzare il tempo necessario ad inoltrare il messaggio m al service provider del servizio descritto dall'advertisement restituito dalla funzione λ. La funzione di accesso è definita in questo modo:

$$r(m) = \begin{cases} n_j & \text{se } n_j \text{ è in contatto} \\ n_w & \text{se } d_{w,j} = \min d_{k,j},\ \forall n_k \in N^i \end{cases}$$

Se il service provider dell'advertisement adv_j è in contatto con n_i, allora invia direttamente il messaggio, altrimenti deve selezionare un nodo con il minimo valore della temporal distance con il service provider.

Se la funzione di selezione non individua alcun advertisement, allora l'algoritmo tenta di inoltrare la query ai nodi in contatto che massimizzano il valore di *Social Centrality* (algoritmo 6, righe 7-8). Un'altra condizione che devono rispettare i nodi in V_q è quella di non aver già ricevuto la stessa query, e quindi non averla già memorizzata all'interno della propria struttura PQ. Tale condizione ha il beneficio di evitare l'inoltro di query a nodi che

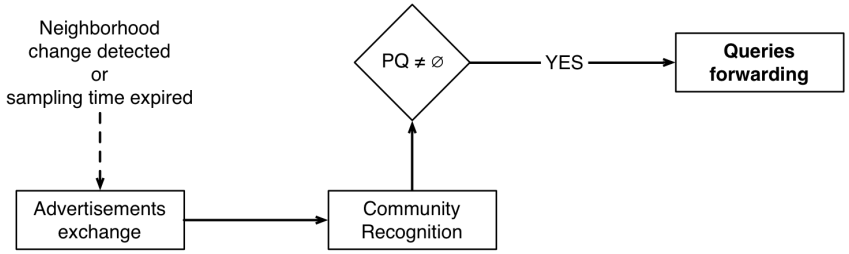


Figura 3.8: Fase proattiva in SIDEMAN$_{PQ}$

sono in attesa della stessa risposta, diminuendo in questo modo il numero di messaggi inviati (vedi 3.2.6).

3.2.5 Componente proattiva

La fase proattiva in SIDEMAN$_{PQ}$ si comporta come mostrato in Figura 3.8. Il nodo n_i controlla periodicamente eventuali cambiamenti all'interno del suo neighborhood (algoritmo 7, riga 1), in particolare quando nuovi nodi entrano a far parte di questo insieme. L'algoritmo 7 descrive sia la propagazione di query, sia quella di advertisement. La fase di scambio degli advertisement è implementata come segue.

Algoritmo 7 ForwardAdvertisementsAndQueries

1: $N^i = neighborhoodDetect()$
2: **for all** $adv \in \mathscr{A}_i$ **do**
3: $\quad forwardAdvertisement(adv)$
4: **end for**
5: $C = \text{RecognizeCommunity}(T^i, N^i, CT^i, \tau)$
6: **for all** $q \in PQ_i$ **do**
7: $\quad V_q = computeSocialCentrality(q, N^i)$
8: $\quad forwardQuery(q, V_q)$
9: **end for**

Un nodo n_i inoltra un advertisement adv al nodo $n_j \in N^i$ se soddisfa le seguenti contdizioni:

- il nodo n_j non possiede già l'advertisement nella sua service cache ($adv \notin \mathscr{A}_j$);
- il nodo n_j è interessato all'advertisement; questo controllo è svolto utilizzando l'indice di Jaccard sull'insieme dei topic dell'advertisement e su quello di n_j, $Jaccard(\mathcal{I}_{adv}, \mathcal{I}_j)$;
- il nodo n_j soddisfa una di queste condizioni:
  1. il service provider in adv appartiene alla comunità di n_j;
  2. il destinatario dell'advertisement adv appartiene alla comunità di n_j.

Oltre a questo tipo di invii, nella fase di scambio degli advertisement, n_i prova ad inoltrare anche tutti quegli advertisement con un particolare destinatario, selezionando i nodi con il miglior valore di temporal distance, nel caso in cui tale destinatario non sia in contatto.

La modalità di invio delle query in questa fase è la stessa utilizzata nel modello reattivo: anche in questo caso sono calcolati i valori di SC sui nodi in contatto e sono scelti i migliori per l'inoltro (algoritmo 7, righe 5-8).

3.2.6 Gestione e ricezione dei messaggi

In questo paragrafo mostriamo la fase di gestione dei messaggi in ingresso. Come abbiamo visto, SIDEMAN$_{PQ}$ utilizza due modalità diverse di invio dei messaggi: quello diretto (nel caso in cui il destinatario è in contatto) e quello indiretto (nel caso in cui il destinatario non è in contatto).

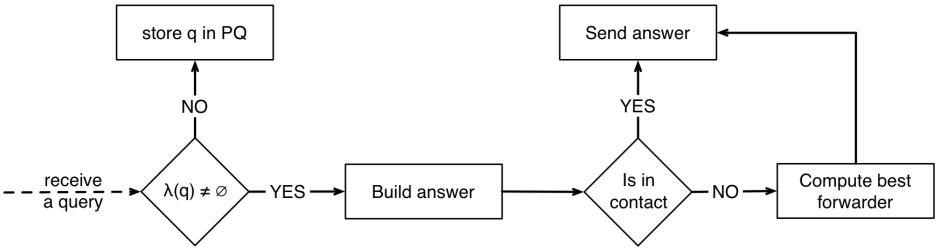


Figura 3.9: Gestione delle query ricevute in SIDEMAN$_{PQ}$

Query Il procedimento di gestione di una query è mostrato in Figura 3.9. Quando il nodo n_i riceve una query q applica la funzione di selezione λ: se $\lambda(q)$ non restituisce alcun advertisement, allora n_i inserisce la query nella struttura dati PQ_i; altrimenti costruisce la risposta con gli advertisement selezionati, e risponde. Se il mittente della query n_q si trova tra i contatti di n_i, allora invia in maniera diretta i messaggi, altrimenti li inoltra al nodo che minimizza la temporal distance verso n_q.

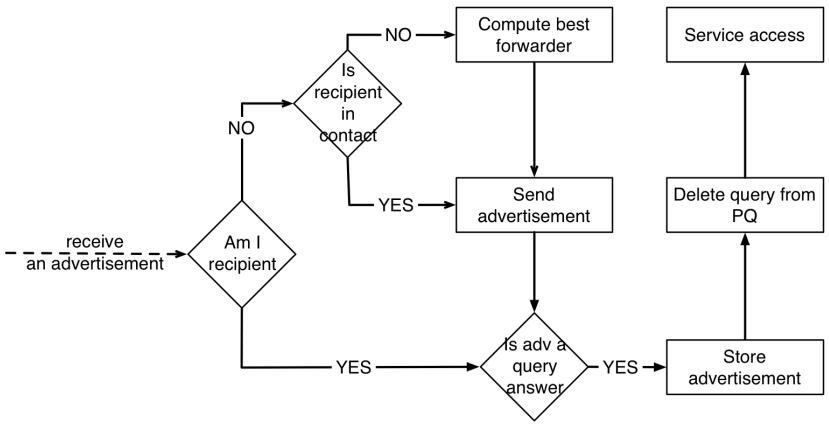


Figura 3.10: Gestione degli advertisement ricevuti in SIDEMAN$_{PQ}$

Advertisement Il procedimento di gestione di un advertisement è mostrato in Figura 3.10. Quando il nodo n_i riceve un advertisement adv, per prima cosa controlla il campo "destinatario" n_{adv}:

- se $n_{adv} = n_i$, allora n_i controlla se nella sua struttura dati PQ_i è presente una query in attesa di risposta che corrisponda all'advertisement ricevuto; in tal caso elimina la query da PQ_i, memorizza adv in cache ed accede al servizio.
- altrimenti n_i inoltra l'advertisement:
  1. al destinatario di adv, se è in contatto,
  2. al nodo $n_k \in N^i$ se n_k minimizza la temporal distance con il destinatario dell'advertisement

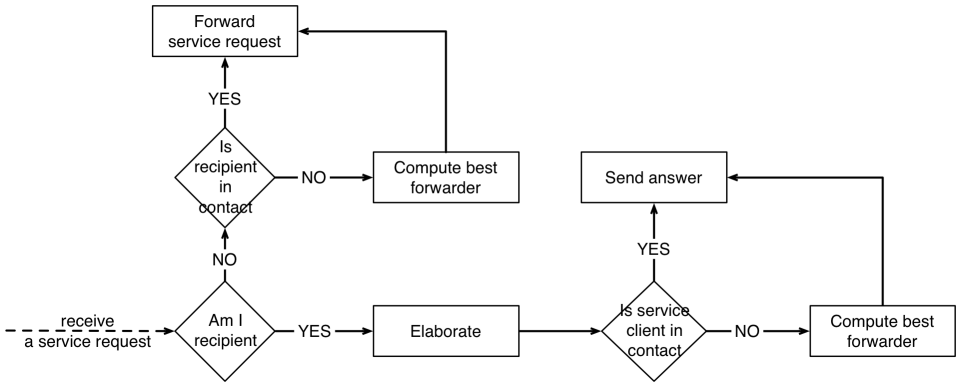


Figura 3.11: Gestione dei messaggi di service request ricevuti in SIDEMAN$_{PQ}$

Service Request Il procedimento di gestione di un messaggio di service request è mostrato in Figura 3.11. Consideriamo un messaggio inviato dal nodo n_k. Se n_i, che ha ricevuto il messaggio ed è il destinatario, ovvero n_i eroga il servizio richiesto, allora elabora la richiesta ed invia la risposta:

- al nodo che ha richiesto il servizio, se è in contatto,
- ad un nodo che minimizza la temporal distance con il service client.

Se, invece, n_i non è il destinatario del messaggio, allora inoltra la richiesta:

- al service provider, se è il contatto,
- ad un nodo che minimizza la temporal distance con il service provider.

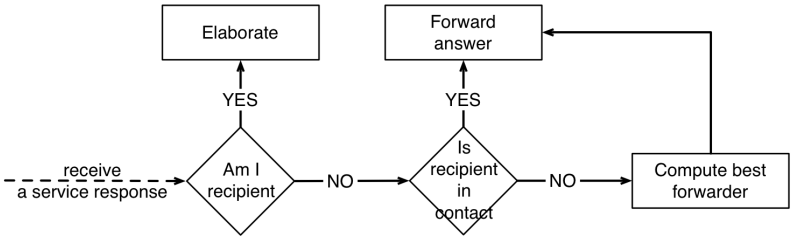


Figura 3.12: Gestione dei messaggi di service response in SIDEMAN$_{PQ}$

Service Response Il procedimento di gestione di un messaggio di service request è mostrato in Figura 3.12. Se n_i, che ha ricevuto il messaggio, è il destinatario, allora elabora la risposta del service provider. Altrimenti deve inoltrare il messaggio:

- al destinatario, se è in contatto con lui,
- ad un nodo che minimizza la temporal distance con il service client.

3.3 Algoritmi di community detection

Le comunità sono uno strumento molto importante nelle MSNs: utilizzando un algoritmo inefficiente o non appropriato SIDEMAN può non funzionare in maniera corretta, e questo può portare ad un utilizzo errato delle risorse messe a disposizione dai nodi o dalla rete. Per questo motivo lo studio degli algoritmi di community detection ha occupato una parte consistente del tempo dedicato alla fase di progettazione di SIDEMAN$_{PQ}$.

Gli algoritmi di community detection hanno il compito di identificare le comunità all'interno di una rete. Esistono due tipologie di algoritmi: quelli centralizzati e quelli distribuiti.

Algoritmi di community detection centralizzati: gli algoritmi centralizzati hanno la necessità di conoscere costantemente la topologia della rete ed i collegamenti che vi sono al suo interno: date queste informazioni è possibile analizzare la rete e determinare le comunità esistenti al suo interno. Uno dei più noti algoritmi centralizzati è quello di Girvan-Newman [GN02]. Per suddividere la rete in un numero ottimo di comunità, questo algoritmo rimuove gli archi con maggior valore di *edge betweeness*, un valore di centralità su grafi $G = (V, E)$. Il valore di *edge betweeness centrality* di un arco $e \in E$ è espresso dalla somma dei rapporti tra il numero di cammini minimi da un nodo n_i ad un nodo n_j che passano per tale arco, che rappresenteremo con $\sigma_{i,j}(e)$, ed il numero totale di cammini minimi che vanno da n_i ad n_j, che

rappresenteremo con $\sigma_{i,j}$; in questo modo la *edge betweeness* è esprimibile come:

$$c_B(e) = \sum_{i,j \in V} \frac{\sigma_{i,j}(e)}{\sigma_{i,j}}$$

Altri approcci centralizzati utilizzano il concetto di *modularity*, che presenteremo in 3.3.1.

Algoritmi di community detection distribuiti: nel caso delle MSNs gli algoritmi distribuiti sono più indicati, poiché non si può assumere una conoscenza globale della topologia della rete, data la mancanza di una struttura centralizzata. In [HYCC07] sono presentati tre algoritmi di community detection distribuiti: SIMPLE, k-CLIQUE e MODULARITY. In SIDEMAN è stato scelto di utilizzare un algoritmo evoluzione di SIMPLE: AD-SIMPLE [BCP11]. AD-SIMPLE ha la caratteristica in più, rispetto agli altri, di essere adattivo; ha infatti la capacità, attraverso un insieme di regole, di eliminare nodi dalle comunità.

Durante lo studio di questi protocolli è stato analizzato anche un'altro algoritmo, che fa parte della famiglia degli algoritmi di clustering distribuiti che utilizzano i concetti di spazio e tempo: DRAFT (Distribuited Rise And Fall spatio-Temporal) [OF13]. Questo algoritmo è adattivo come AD-SIMPLE, ma ha dei vantaggi: *(i)* non ha problemi di permanenza di alcuni nodi obsoleti all'interno delle comunità (cosa che invece è stata riscontrata in AD-SIMPLE), ed *(ii)* è molto più semplice da configurare. Per questi motivi è stato scelto DRAFT nella fase di testing di SIDEMAN$_{PQ}$.

Presenteremo nei prossimi paragrafi gli algoritmi studiati durante la fase di progettazione: SIMPLE, k-CLIQUE, MODULARITY, gli algoritmi base per community detection in ambiente distribuito, AD-SIMPLE, l'algoritmo inizialmente utilizzato nella sperimentazione di SIDEMAN, ed infine DRAFT, l'algoritmo utilizzato per la sperimentazione di SIDEMAN$_{PQ}$.

3.3.1 SIMPLE, k-CLIQUE e MODULARITY

Utilizzeremo una comune terminologia per questi primi tre algoritmi che presenteremo; le strutture dati più importanti utilizzate in questi algoritmi sono:

- **Familiar Set (F):** il familiar set di un nodo n_i è formato da tutti quei nodi che n_i ha incontrato, e con i quali è stato in contatto per un tempo complessivo[2] maggiore di una certa soglia prestabilita t_{th}.

[2]Chiameremo il tempo di contatto complessivo *cumulative contact duration*, denotato come t_{cum}

- **Local Community (C):** la local community di un nodo n_i è formata da tutti i nodi appartenenti al suo familiar set, insieme ad altri nodi che rispettano certe proprietà, specifiche per gli algoritmi che andremo ad analizzare.

Notazione	
F_i	familiar set del nodo n_i
C_i	local community del nodo n_i
$\hat{F}_j$	stima di n_i del familiar set del nodo n_j
$FSoLC_i$	stima di n_i dei familiar set di tutti i nodi appartenenti a C_i ($FSoLC = \{\hat{F}_j \mid n_j \in C_i\}$)

Tabella 3.3: Notazione usata in SIMPLE, k-CLIQUE e MODULARITY

In tabella 3.3 troviamo un resoconto della notazione utilizzata nel resto del paragrafo.
In questi algoritmi, quando due nodi si incontrano, per prima cosa si scambiano le loro informazioni locali, ed ognuno decide se e chi includere nel proprio familiar set (*threshold criteria*), oppure nella propria local community (*admission criteria*). In più, ogni nodo valuta se la sua local community deve essere unita, in maniera parziale o completa, con quella del nodo incontrato (*merging criteria*).
La prima regola (*threshold criteria*) è la stessa in tutti e tre gli algoritmi.

Threshold Criteria: quando due nodi si incontrano, ogniuno aggiorna il proprio *cumulative contact duration*, t_{cum}. Se $t_{cum} \geq t_{th}$, allora il nodo incontrato viene aggiunto nel familiar set (e di conseguenza anche nella local community).

Le altre due regole sono specifiche per algoritmo.

SIMPLE In questo algoritmo i nodi si scambiano pochissime informazioni. Supponiamo un incontro tra i nodi n_i ed n_j: durante il contatto i due nodi si scambiano le proprie strutture dati: la local community ed il familiar set. Ricevute queste informazioni, valutano le seguenti regole (consideriamo n_i):

- **Admission Criteria:** in caso di fallimento del *threshold criteria*, se il numero di nodi condivisi tra C_i ed F_j è maggiore di λ volte il numero di nodi in F_i, allora n_j è aggiunto alla local community di n_i:

$$|C_i \cap F_j| > \lambda \cdot |F_i| \Rightarrow C_i = C_i \cup \{n_j\}$$

- **Merging Criteria:** nel caso in cui n_j sia stato aggiunto alla local community di n_i come conseguenza di una delle due regole precedenti,

se il numero di nodi condivisi tra C_i e C_j è maggiore γ volte il numero di nodi dell'unione di C_i e C_j, allora le due local community sono unite:

$$|C_i \cap C_j| > \gamma \cdot |C_i \cup C_j| \Rightarrow C_i = C_i \cup C_j$$

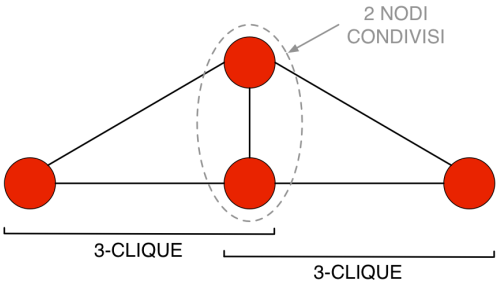


Figura 3.13: Esempio di k-CLIQUE con $k = 3$

k-CLIQUE Questo è la versione distribuita dell'algoritmo presentato in [PDFV05]. Una comunità è definita come l'unione di tutti i k-clique che possono essere raggiunti da ogniuno degli adiacenti k-cliques, dove un k-clique è un grafo completo di dimensione k: due k-cliques sono adiacenti se condividono esattamente $k-1$ nodi (vedi Figura 3.13). Oltre al familiar set ed alla local community, quando i nodi n_i ed n_j si incontrano, scambiano anche l'approssimazione del familiar set di tutti i nodi della local community ($FSoLC$). I nodi quindi valutano le seguenti regole (consideriamo come prima n_i):

- **Admission Criteria:** in caso di fallimento del *threshold criteria*, se il familiar set di n_j contiene almeno $k-1$ nodi della local community di n_i, allora n_j è aggiunto alla local community di n_i:

$$|C_i \cap F_j| \geq k-1 \Rightarrow C_i = C_i \cup \{v_j\}$$

- **Merging Criteria:** nel caso in cui n_j sia stato aggiunto alla local community di n_i come conseguenza di una delle due regole precedenti, se il familiar set di ogni nodo all'interno della local community di n_j (ad esempio n_w) contiene almeno $k-1$ nodi della local community di n_i, allora n_w è aggiunto alla local community di n_i:

$$|C_i \cap \hat{F}_w| \geq k-1 \Rightarrow C_i = C_i \cup \{n_w\} \text{ , dove : } n_w \in C_j \text{ e } \hat{F}_w \in FSoLC_j$$

Nel caso in cui la regola del *merging criteria* fosse soddisfatta, sarebbe necessario aggiornare la $FSoLC$ ($FSoLC_i = FSoLC_i \cup \hat{F}_j$).

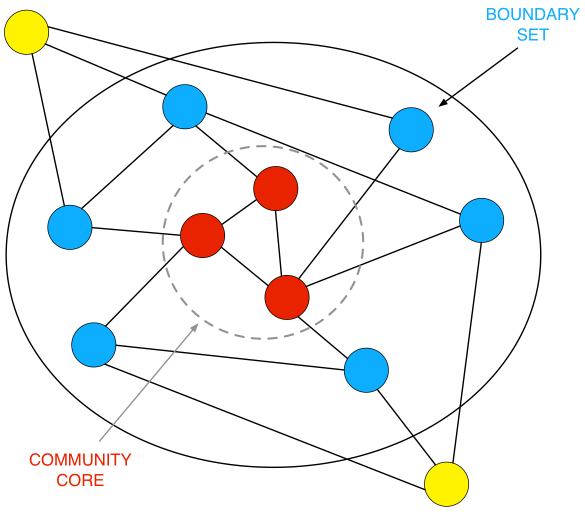


Figura 3.14: Esempio di *boundary set* in MODULARITY

MODULARITY Questo algoritmo si basa sul concetto di *local modularity* (R) e sul calcolo del suo tasso di variazione (ΔR) quando viene aggiunto un nuovo nodo alla comunità esistente.

Per definire la local modularity è necessario introdurre in concetto di *boundary set*. Il boundary set di un nodo è definito come il sottoinsieme di nodi, nella sua local community, con archi uscenti dalla comunità che li connettono a nodi esterni (Figura 3.14).

Tenendo in considerazione questa definizione, possiamo definire la local modularity come:

$$R = \frac{I}{|T|}$$

dove I è il numero di archi che non terminano in nodi esterni alla local community, mentre T è l'insieme degli archi con un estremo su nodi del boundary set. Se quest'ultimo coincide con la local community, allora $R = 1$ per definizione.

Quando un nuovo nodo n_j è aggiunto alla local community C_i del nodo n_i con buondary set B_i, la variazione della local modularity è calcolata in questo modo:

$$\Delta R_i = \frac{x - R_i \cdot y - z \cdot (1 - R_i)}{|T| - z + y}$$

dove x è il numero di archi in T che terminano in n_j, y è il numero di archi che saranno aggiunti a T con l'inclusione di n_j e z è il nuemero di archi che saranno rimossi da T con l'inclusione di n_j.

L'algoritmo è il seguente: quando due nodi si incontrano si scambiano: *(i)* F, *(ii)* C e *(iii)* $FSoLC$. Dopo lo scambio entrambi aggiornano gli insiemi, e valutano le seguenti regole (consideriamo sempre n_i):

- **Admission Criteria:** in caso di fallimento del *threshold criteria*, se la differenza tra la local modularity misurata prima e dopo aver incluso n_j nella local community supera 0, allora il nodo incontrato n_j è inserito nella local community di n_i:

$$\Delta R_i \geq 0 \Rightarrow C_i = C_i \cup \{n_j\}$$

- **Merging Criteria:** nel caso in cui n_j sia stato aggiunto alla local community di n_i come conseguenza di una delle due regole precedenti, l'algoritmo valuta se includere in C_i i nodi dell'insieme K:

$$K = \{n_k | \exists w \text{ t.c. } n_w \in C_i \cap C_j \wedge n_k \in \hat{F}_w \wedge n_k \in C_j \setminus C_i\}$$

  L'insieme K è formato da un sottoinsieme dei nodi di C_j che sono adiacenti ai nodi condivisi tra C_i e C_j.

  Per ogni nodo $n_k \in K$, se il familiar set di n_k ùn sottoinsieme della local community di n_i, allora n_k è aggiunto alla local community di n_i

$$\hat{F}_k \subseteq C_i \Rightarrow C_i = C_i \cup \{n_k\}$$

  In più, se la variazione della local community ΔR_i è calcolata per tutti i restanti nodi di K. Nodi con $\Delta R_i > 0$ sono aggiunti alla local community di n_i. Per tutti quei nodi con $\Delta R_i \leq 0$, il valore di ΔR_i è ricalcolato e ricontrollato dopo ogni aggiunta. Questa procedura può essere ripetuta più volte, e termina quando *(i)* K è vuoto, oppure *(ii)* dopo aver aggiunto n_k a C_i, e $\Delta R_i \leq 0$ per tutti i rimanenti nodi in K. Inoltre, ad ogni aggiunta, $\hat{F}_k$ è unito a $FSoLC_i$.

Considerazioni su SIMPLE, k-CLIQUE e MODULARITY Le soglie t_{th}, λ, γ e k sono dei parametri di design, utilizzati per variare il comportamento degli algoritmi, in modo da calibrare il sistema di community detection. Questi algoritmi, come abbiamo visto, richiedono ogniuno una diversa quantità di memoria per il salvataggio delle strutture di supporto; inoltre richiedono ogniuno un diverso consumo di energia (in termini di numero di messaggi da inviare). SIMPLE è l'algoritmo più semplice, e che utilizza meno risorse, mentre MODULARITY è quello più complesso, sia in termini di memoria necessaria, sia in termini di quantità di elaborazione necessaria per mantenere consistenti le strutture di supporto all'algoritmo. k-CLIQUE rappresenta una soluzione intermedia, che utilizza la stessa quantità di memoria di MODULARITY, ma una minore complessità nel calcolo.

Questo tipo di algoritmi hanno un problema comune: i nodi che li eseguono non hanno una visione consistente del proprio familiar set o della propria local community, dato che mantengono in memoria tutti i nodi incontrati senza implementare meccanismi di rimozione nel tempo.
Per questo motivo sono stati introdotti degli algoritmi (come AD-SIMPLE e maggiormente DRAFT) che rappresentano meglio il comportamento dinamico e sociale degli utenti.

3.3.2 AD-SIMPLE

AD-SIMPLE estende SIMPLE con due procedure:

- dei meccanismi per comprendere la datazione dei contatti tra gli utenti;
- la cancellazione dei nodi dalle comunità.

AD-SIMPLE (**Adaptive Detection SIMPLE**) è stato introdotto in [BCP11]: questo algoritmo è in grado di adattare le comunità al cambiamento che subiscono nel tempo. L'idea principale dell'algoritmo è quella di mantenere lo stesso funzionamento originale di SIMPLE, per quanto riguardo l'inclusione di nodi nei familiar set e nelle local community, introducendo in più delle regole (*familiar set pruning policy* e *local community pruning policy*) per identificare ed eliminare dalle comunità nodi che non sono più in contatto da molto tempo.

Familiar set pruning policy L'idea di questa regola è quella di analizzare la percentuale di contatto, su tutto il tempo di esecuzione, con i vari nodi del familiar set: in base a questo rapporto si decide se eliminare o no un nodo dall'insieme (si utilizza anche in questo caso un valore di soglia). Per

Figura 3.15: AD-SIMPLE: $Sample\Delta T$ è dato dal rapporto tra il tempo in cui due nodi sono stati in contatto nel time frame (C) ed la durata dell'interno time frame (T): $Sample\Delta T = \frac{C}{T}$

questo motivo il tempo è diviso logicamente in slots (di durata T) e i nodi calcolano la percentuale della durata del contatto in ogni slot, per tutti i nodi del familiar set ($Sampe\Delta T$ - Figura 3.15). Alla fine dello slot, il nodo calcola $Estimated\Delta T$ come la media pesata tra la stima precedente a questa nuova misurazione:

$$Estimated\Delta T = \alpha \cdot Estimated\Delta T + (1 - \alpha) \cdot Sample\Delta T$$

Parametri algoritmi (AD-) SIMPLE	
SIMPLE	
t_{th}	valore di soglia utilizzato per *threshold criteria*
λ	valore di soglia utilizzato per *admission criteria*
γ	valore di soglia utilizzato per *merging criteria*
AD-SIMPLE	
α	parametro per la variabilità del familiar set
$FSout_{th}$	valore di soglia utilizzato nel *familiar set pruning policy*
$LCout_{th}$	valore di soglia utilizzato nel *local community pruning policy*

Tabella 3.4: Parametri di AD-SIMPLE

dove α è il parametro che da maggiore o minore peso al valore di $Estimated\Delta T$ rispetto a $Sample\Delta T$ nel calcolo. Un valore "piccolo" di α, infatti, porta a variazioni di $Estimated\Delta T$ più rapide, mentre un valore "grande" di α mantiene più stabile tale valore.

Se un nodo n_j, alla fine del time slot T, ha un valore di $Estimated\Delta T_i$ tale che:

$$Estimated\Delta T_i < FSout_{th}$$

allora è eliminato dal familiar set; $FSout_{th}$ è la soglia minima affinché un nodo sia mantenuto all'interno dell'insieme.

Local community pruning policy L'idea utilizzata in questa regola invece è quella di utilizzare un timer ($LCout_{th}$) per ogni nodo della local community. Il timer è inizializzato all'ingresso di un nodo nella local community (ad esempio quando n_j è inserito in C_i), ed è ripristinato al valore iniziale quando:

1. n_i incontra direttamente n_j;
2. n_i incontra un'altro nodo che contiene n_j nella propria local community.

Un nodo è eliminato dalla local community quando il suo timer termina; se il nodo si trova anche nel familiar set, allora viene eliminato anche da questo.

Conclusioni su AD-SIMPLE AD-SIMPLE si è rivelato un algoritmo funzionante per i nostri propositi: tuttavia non è molto semplice da utilizzare, dato il gran numero di parametri da inizializzare (tabella 3.4). Tale difficoltà è stata incontrata nella fase di sperimentazione con tracce di mobilità molto diverse tra loro, che avevano quindi bisogno di utilizzare parametri diversi. Per questo motivo è stato preferito l'algoritmo di clustering DRAFT, più adattabile ai diversi scenari analizzati.

3.3.3 DRAFT

Questo algoritmo si basa sulla stessa idea di AD-SIMPLE, ma riesce a rispecchiare in maniera più verosimile la struttura delle comunità all'interno della rete; inoltre il minor numero di parametri lo rende maggiormente configurabile. L'algoritmo applica una funzione di degrado ad intervalli di tempo definiti, per eliminare dalle comunità nodi obsoleti.

L'algoritmo utilizza il tempo di contatto cumulativo, o *cumulative contact time*, come indice per mantenere o eliminare un nodo da una comunità: ci riferiremo a questo valore con t_{cum}. DRAFT utilizza tre parametri per gestire l'inclusione (o l'esclusione) dei nodi della local community:

1. il *familiar threshold* τ, la soglia di t_{cum} oltre la quale un nodo è inserito nella comunità;
2. un *time frame* t, l'intervallo di tempo oltre il quale il t_{cum} subisce una diminuzione;
3. il *decay ratio* δ, un valore nell'intervallo $0 \leq \delta \leq 1$, che dice di quanto il tempo cumulativo di contatto totale viene ridotto ad ogni *time frame*.

In questo modo i nodi che non entrano in contatto per un certo periodo sono eliminati dalla local community.

Nel processo di community detection sono utilizzate tre strutture dati di supporto; un nodo n_i mantiene:

1. **Neighbour set (N_i):** un insieme di coppie $< n_j; N_{ij} >$: n_j è il nodo incontrato da n_i, mentre N_{ij} è il tempo di contatto cumulativo tra il nodo n_i ed il nodo n_j;
2. **Local community (C_i):** un insieme che contiene i nodi che fanno parte della comunità;
3. **Deletion table (D_i):** un insieme che contiene i nodi selezionati per l'eliminazione dalla local community, e quelli che sono già stati eliminati.

Il procedimento con il quale vengono create le comunità è il seguente:

Regola 1 Inizialmente C_i è formato dal solo $\{n_i\}$, mentre N_i e D_i sono vuoti (algoritmo 8).

Algoritmo 8 *Regola 1*

1: $C_i = \{n_i\}$
2: $N_i = \varnothing$
3: $D_i = \varnothing$

Regola 2 Quando n_i incontra n_j, lo inserisce in N_i se non vi si trova già, ed incrementa N_{ij}, il contatore di t_{cum} per n_j (algoritmo 9, righe 1-2).

Regola 3 Se N_{ij} supera la soglia τ, allora n_i richiede delle informazioni del contatto n_j (C_j e D_j); se la richiesta ha successo:

1. n_j viene aggiunto a C_i (algoritmo 9, righe 7-9)
2. se n_j era stato selezionato in precedenza per la cancellazione, quindi si trova in D_i, è cancellato dall'insieme (algoritmo 9, righe 4-6).

Algoritmo 9 *Regole 2-3*

```
1: N_i = N_i ∪ {n_j}
2: N_ij = N_ij + 1
3: if N_ij ≥ τ then
4:     if n_j ∈ D_i then
5:         D_i = D_i \ {n_j}
6:     end if
7:     if n_j ∉ C_i then
8:         C_i = C_i ∪ {n_j}
9:     end if
10: end if
```

Regola 4 Una volta che le informazioni richieste nella *Regola 3* sono state ricevute ed elaborate, DRAFT cerca di eliminare alcuni nodi obsoleti:

1. n_i controlla i nodi presenti in D_i ed in D_j. Trattandosi di incontri opportunistici, è possibile che in questi insieme vi siano molti nodi in comune. Tutti questi nodi sono eliminati da entrambe le coppie di insiemi (algoritmo 10, righe 2-5).
2. se un nodo in D_i si trova in C_j ma non in D_j allora il nodo è eliminato da D_i (algoritmo 10, righe 6-8).
3. se un nodo si trova in D_i, ma non in C_j o D_j allora il nodo è lasciato all'interno di D_i.

Algoritmo 10 *Regola 4*

```
1: if n_j ∈ C_i then
2:     for all n_k ∈ D_i ∩ D_j do
3:         D_i = D_i \ {n_k} ; D_j = D_j \ {n_k} ;
4:         C_i = C_i \ {n_k} ; C_j = C_j \ {n_k} ;
5:     end for
6:     for all n_k ∈ D_i ∩ C_j do
7:         D_i = D_i \ {n_k}
8:     end for
9: end if
```

Regola 5 Alla fine di ogni time frame:

1. ogni nodo presente allo stesso tempo in C_i ed in D_i è ritenuto obsoleto e rimosso da C_i (algoritmo 11, righe 1-5);
2. ogni elemento di N_i è moltiplicato per il fattore di decremento δ, in modo da aggiornare i tempi. Tutti i nodi per cui $N_{ij} < \tau$ vengono segnalati per la cancellazione ed inseriti in D_i, per essere cancellati alla fine del successivo time frame (algoritmo 11, righe 6-11).

Algoritmo 11 *Regola 5*

```
1: for all n_j ∈ D_i do
2:     if n_j ∈ C_i then
3:         C_i = C_i \ {n_j}
4:     end if
5: end for
6: for all n_j ∈ N_i do
7:     N_ij = N_ij · δ
8:     if  N_ij < τ ∧ n_j ∈ C_i  then
9:         D_i = D_i ∪ {n_j}
10:    end if
11: end for
```

Conclusioni su DRAFT L'algoritmo presentato ha le caratteristiche adatte al nostro utilizzo. Come già detto, l'algoritmo di community detection è una parte importante per SIDEMAN e SIDEMAN$_{PQ}$: la selezione dei nodi a cui inviare le richieste è svolta, in questi due algoritmi, considerando i nodi in contatto, ma che allo stesso tempo si trovano nella local community del mittente. Per questo è importante che le comunità siano riconosciute in maniera precisa.

Un'altro aspetto importante da considerare è la semplicità di DRAFT. I

valori di τ, t e γ dipendono infatti dalla mobilità dei partecipanti, ed al livello di reattività richiesto alle comunità. Ad esempio, consideriamo τ: data la natura degli incontri dell'uomo, dinamici e diurni, la lunghezza di τ dovrebbe essere, in applicazioni reali, maggiore del valore medio della durata degli incontri, ma allo stesso tempo inferiore a 24 ore.

La possibilità di modificare in maniera semplice l'andamento dell'algoritmo, con soltanto tre parametri da variare, è la caratteristica che rende questo algoritmo più fruibile per questo lavoro di tesi.

Capitolo 4

Implementazione e valutazione

In questo capitolo descriveremo la fase di implementazione e valutazione dell'algoritmo SIDEMAN$_{PQ}$.
Per prima cosa affronteremo la fase di studio delle tracce di mobilità analizzate, mostrandone le caratteristiche principali. Infatti una traccia di mobilità può essere studiata sotto diversi aspetti sia temporali che sociali. Ad esempio, alcune metriche che descrivono i parametri temporali sono la durata dei contatti tra i nodi ed i tempi di intercontatto, mentre quelle che descrivono gli aspetti sociali sono la dimensione delle comunità e quella del neighborhood. Questi aspetti sono utili per comprendere quale potrebbe essere l'andamento dell'algoritmo di service discovery: tracce con maggiore densità nei contatti dovrebbero dare risultati migliori rispetto a tracce con densità minore, dato che garantiscono un maggior numero di opportunità per lo scambio di informazioni. Vedremo in 4.3 che quest'ultima affermazione non è sempre valida.

Successivamente descriveremo lo scenario applicativo di riferimento, ed il simulatore ONE [KKO10], utilizzato in fase di sviluppo e test di SIDEMAN$_{PQ}$. ONE è un simulatore modulare sviluppato in Java, che mette a disposizione un gran numero di strumenti per le fasi di sviluppo e test di protocolli su Delay Tolerant Networks, e quindi utilizzabile anche per le MSNs. Utilizzando i tool messi a disposizione da questo simulatore è stato possibile implementare e valutare il comportamento di SIDEMAN$_{PQ}$.

Infine mostreremo le metriche di interesse per i protocolli di service discovery su MSN, e le performance di SIDEMAN$_{PQ}$ ottenute dalla campagna di simulazioni preliminari.

4.1 Le tracce di mobilità

Le tracce di mobilità sono la parte più importante dello scenario simulativo. Una traccia di mobilità definisce i contatti tra i nodi presenti nella rete. Con

il termine contatto si intende l'opportunità, tra una coppia di nodi, di scambiare messaggi ed informazioni. Le tracce di mobilità possono essere di due tipi: geografiche o di collocazione. Quelle geografiche sono descritte da un insieme di coordinate, corrispondenti alla posizione dei nodi, e da parametri di mobilità dei nodi (velocità, direzione degli spostamenti). Le tracce di mobilità di collocazione, invece, descrivono i contatti, indicando il loro inizio e la loro fine. Durante la fase di analisi delle tracce di mobilità sono state utilizzate quest'ultimo tipo, nelle quali i contatti sono rilevati dalle interfacce Bluetooth o WiFi presenti nei dispositivi utilizzati per la realizzazione di questi datasets. Durante lo svolgimento della tesi sono stati studiati principalmente i dataset Infocom5, Cambridge [SGC+06] e Reality MIT [EP05] (tabella 4.1).

	Infocom5	**Cambridge**	**Reality MIT**
Ambiente	Conferenza	Campus	
Durata (giorni)	3	12	246
Numero di nodi	41	36	97
Tipo di dispositivi	iMote		Phone
Numero di incontri	22459	10641	102594
Daily encounter probability	0.78	0.24	0.01
Granularità (secondi)	120	600	300
Locazione grografica	No		ID cella

Tabella 4.1: Caratteristiche di Infocom5, Cambridge e Reality. Il numero di incontri tra i partecipanti è stato ottenuto attraverso i log delle connessioni Bluetooth tra i dispositivi. La *Daily encounter probability* è la probabilità che l'incontro con un dato nodo avvenga in un dato giorno, e la granularità è il tempo tra due fasi di discovery dei vicini.

Descriveremo brevemente le tracce, per poi illustrare le loro caratteristiche sociali e temporali nel paragrafo 4.1.2.

Infocom5 La traccia di mobilità Infocom5 [SGC+06] è stata creata durante la sperimentazione sulla mobilità umana svolta nella la conferenza Infocom 2005, da parte di un gruppo di ricercatori e studenti di dottorato dell'università di Cambridge. Per la raccolta dei dati sono stati utilizzati degli Intel iMotes, dei piccoli dispositivi composti da un processore ARM, un'interfaccia Bluetooth, una memoria flash ed una batteria. Una volta distribuiti tra i 41 partecipanti, questi dispositivi hanno eseguito, ogni 2 minuti, la scansione dei dispositivi vicini, creando, in questo modo, dei log contenenti gli incontri.

Alla conferenza erano presenti, oltre ai partecipanti alla sperimentazione, anche altri individui, e di conseguenza altri dispositivi: la traccia utilizzata nel simulatore è stata ottenuta da un'elaborazione a posteriori dei dati: sono

stati infatti eliminati dal dataset tutti i contatti avvenuti con indirizzi MAC diversi da quelli degli iMotes.

Nella sperimentazione di SIDEMAN$_{PQ}$ questa traccia rappresenta l'esempio di test nel contesto di una conferenza, quindi in un ambiente insolito rispetto alla quotidianità: le caratteristiche di Infocom5 hanno permesso di valutare SIDEMAN$_{PQ}$ in uno scenario diverso da quelli presentati in Cambridge e Reality MIT.

Cambridge La traccia di mobilità Cambridge appartiene allo stesso insieme di sperimentazioni svolte in [SGC+06] dagli stessi ideatori di Infocom5. Anche in questo caso sono stati utilizzati degli iMotes per la raccolta dei dati. Questi dispositivi sono stati distribuiti a due gruppi di utenti: il gruppo dell'Intel Research Cambridge Laboratory, per sei giorni, e successivamente al gruppo del Computer Laboratory presso l'università di Cambridge, per altri sei giorni: la traccia ha quindi una durata totale di 12 giorni, ed ha coinvolto un totale di 36 utenti. Trattandosi di una sperimentazione di maggiore durata, è stato utilizzato un tempo di campionamento maggiore rispetto a quello di Infocom5, passando da 2 minuti a 10 minuti, in modo da evitare un degrado troppo rapido delle batterie degli iMotes. Questa traccia rappresenta uno dei due esempi di test ambientati in un campus; possiamo definire Cambridge come una traccia intermedia tra Infocom5 e Reality MIT.

Reality MIT La traccia di mobilità Reality MIT è la più completa dal punto di vista delle informazioni fornite. La sperimentazione, svolta attraverso l'utilizzo di smartphones Nokia 6600 con preinstallate applicazioni per la raccolta di dati, ha fornito, oltre alle informazioni di prossimità dei contatti raccolte attraverso Bluetooth e WiFi, anche:

- informazioni reperite dai sensori presenti sugli smarphones, come microfoni, accelerometri e GPS;
- log di chiamate effettuate/ricevute e di messaggi inviati/ricevuti;
- log dell'utilizzo del dispositivo e delle applicazioni al suo interno.

La sperimentazione ha visto la partecipazione di 97 utenti, alcuni dal MIT Media Laboratory, altri dalla MIT Sloan business school. Il tempo di campionamento in Reality MIT è di 300 secondi (5 minuti). L'utilizzo di smarphones ha garantito la possibilità di eseguire la raccolta dei dati per un periodo più grande e su un campione di utenti più ampio, in modo da studiare il comportamento dei partecipanti in maniera più accurata. Nella nostra analisi utilizzeremo una porzione della traccia di 20 giorni.

4.1.1 Metriche per l'analisi delle tracce

In questo paragrafo vengono riportate le pricipali caratteristiche sociali e temporali utilizzate per l'analisi dei datasets.

Le caratteristiche sociali di una traccia di mobilità sono descritte attraverso lo studio dei risultati dell'algoritmo di community detection DRAFT:

- le comunità riconosciute, ed in particolare la dimensione media di questo insieme di nodi;
- il neighborhood, in particolare la dimensione media di questo insieme di nodi.

Le comunità rappresentano l'insieme di nodi che, secondo l'algoritmo di community detection, hanno un particolare legame. Il neighborhood è invece l'insieme dei nodi in contatto, con i quali è possibile stabilire una connessione.

In SIDEMAN$_{PQ}$ questo tipo di caratteristiche influenza il comportamento dell'algoritmo. L'inoltro dei messaggi, infatti, considera:

- le comunità dei nodi, ad esempio nel calcolo della *social centrality*, per individuare i migliori candidati nell'inoltro delle query;
- il nighborhood, per scambiare gli advertisement durante la fase proattiva dell'algoritmo.

Le caratteristiche temporali di una comunità sono date dalle seguenti caratteristiche:

- il numero medio di contatti in ogni ora (per nodo);
- la durata media dei contatti;
- il numero medio di incontri e di contatti singoli;
- la distribuzione complementare (CCDF) dei tempi di intercontatto.

Queste caratteristiche descrivono *(i)* la distribuzione dei contatti tra i nodi durante le ore della giornata, *(ii)* il tempo medio trascorso tra l'inizio e la fine di un contatto, *(iii)* il rapporto tra il numero totale di contatti ed il numero di nodi incontrati, e *(iv)* l'andamento dei tempi di intercontatto.

4.1.2 Analisi delle tracce

Mostreremo adesso l'analisi delle tracce secondo le metriche appena descritte, distinguendo tra gli aspetti di tipo sociale e quelli di tipo temporale.

Analisi delle caratteristiche sociali Figura 4.1 mostra la dimensione media della comunità e quella del neighborhood: come possiamo vedere, in Infocom5 sono presenti delle comunità molto più grandi rispetto alle altre due tracce, soprattutto rispetto a Reality MIT. Questo aspetto dipende dall'ambiente in cui è stata svolta la raccolta dei dati: trattandosi, in Infocom5, di una conferenza, molti nodi si sono trovati molto spesso negli stessi luoghi. La stessa cosa non succede in Cambridge ed in Reality MIT: l'ambiente in cui sono stati raccolti i dati per queste tracce è quello di un campus, nello specifico di alcuni laboratori, dove le caratteristiche temporali dei contatti sono molto diverse rispetto al caso di una conferenza. L'andamento oscillante della dimensione di comunità e neighborhood segue un andamento giornaliero: infatti, durante la giornata, un nodo ha maggiore probabilità di incontrare altri nodi e di passare tempo con loro, rispetto alla notte, dove solitamente non sono stabiliti contatti.

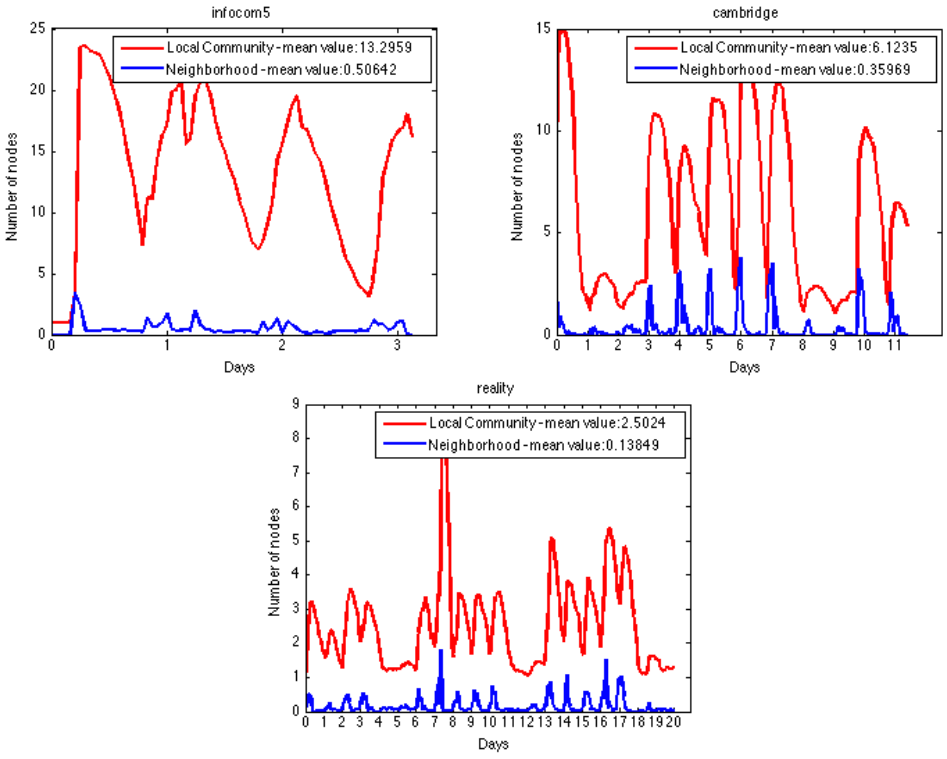


Figura 4.1: Comunità e Neighborhood

Durante questa fase di analisi è stato interessante studiare anche il comportamento di alcuni nodi nelle diverse tracce; in Figura 4.2 è mostrato l'andamento delle comunità di alcuni nodi selezionati all'interno della popolazione. In Infocom5 ed in Cambridge è stata analizzata la comunità del nodo con identificativo 28, mentre in Reality MIT quello con identificativo

56. La linea blu identifica il nodo analizzato, mentre il colore rosso indica che il nodo, il cui identificativo corrisponde al valore sull'ascissa, si trova nella comunità del nodo monitorato. I grafici in Figura 4.2 mostrano tre comportamenti completamente diversi. Nel caso di Infocom5, il nodo monitorato è in contatto con molti dei nodi; questo aspetto è indice della densità di tale scenario. In Cambridge risalta la ciclicità della presenza in comunità di un numero molto più ristretto di nodi rispetto ad Infocom5; in questo scenario si nota anche la totale mancanza di contatti durante alcuni giorni, dovuta al fatto che i partecipanti alla sperimentazione, durante il fine settimana, non fossero in contatto. Anche Reality MIT ha un andamento ciclico simile a quello riscontrato in Cambridge, con la differenza che sono presenti alcuni nodi, come il nodo 28 ed il nodo 85, che fanno parte della comunità del nodo monitorato per lunghi periodi, ed in maniera continua; questo potrebbe far pensare ad individui che lavorano insieme, ma che hanno dei contatti anche all'esterno del campus.

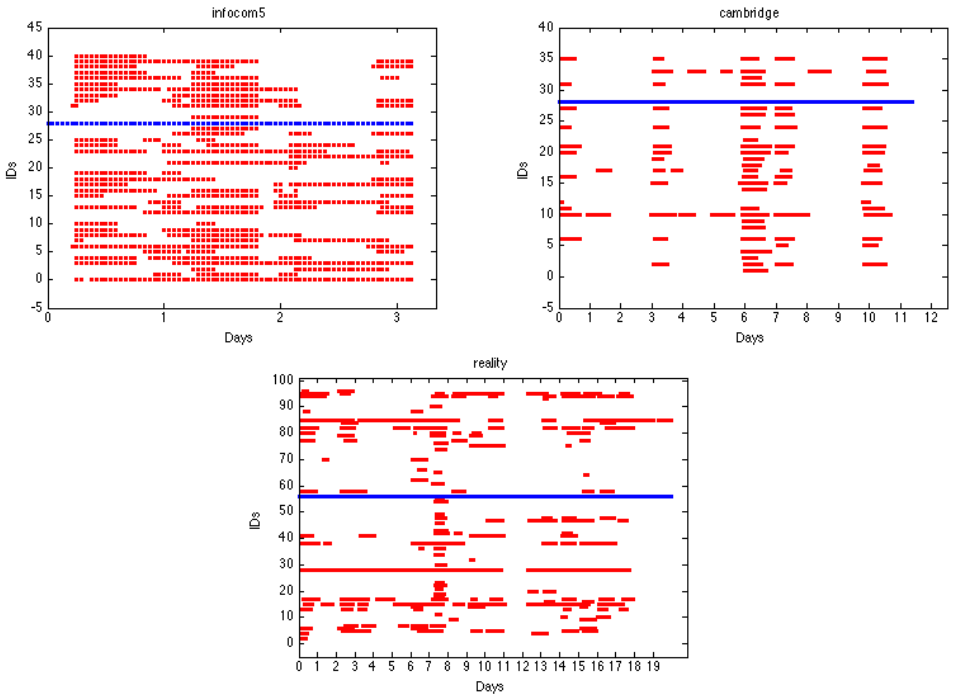


Figura 4.2: Comunità di un nodo

Dall'analisi dei grafici riportati finora risulta che la traccia Infocom5 ha comunità di dimensione maggiore; aumentando l'ampiezza dell'ambiente di campionamento, passando a Cambridge, fino a Reality MIT, notiamo una diminuzione evidente negli insiemi delle comunità.

Analisi delle caratteristiche temporali Il grafico del **numero medio di contatti in ogni ora** nei tre diversi scenari è mostrato in Figura 4.3.

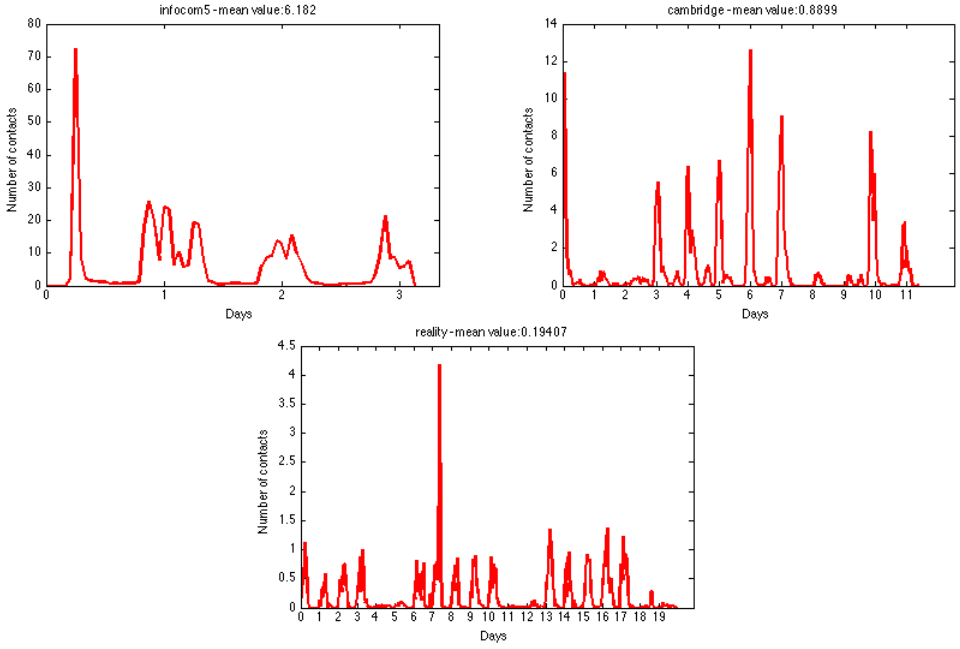


Figura 4.3: Numero medio di contatto in ogni ora

Come possiamo vedere, questo grafico mostra l'aspetto di ciclicità delle tracce: il numero di contatti cresce durante le ore diurne, per poi decrescere nelle notturne. Infocom5 è la traccia di mobilità con il maggior numero di contatti, mentre Reality MIT quella con il minor numero. Cambridge, come già anticipato, è una traccia intermedia: non raggiunge la quantità di incontri di Infocom5, ma, durante le ore diurne, arriva ad un numero di contatti che va dai 6 ai 12 nodi. Per SIDEMAN$_{PQ}$ il numero di contatti è un aspetto molto importante, dato che maggiore è il numero di contatti, maggiori saranno le opportunità di scambio dei messaggi: di conseguenza aumenteranno anche le porbabilità di inoltrare query ed advertisement.

Allo stesso tempo dobbiamo considerare i valori della **durata media dei contatti** nei tre diversi scenari, mostrato in Figura 4.4. Come possiamo vedere, in Infocom5 troviamo un andamento più costante rispetto agli altri due scenari. Questo è sempre dovuto all'ambiente di raccolta dei dati. L'andamento mostrato in figura indica che, nonostante in Cambridge ed in Reality MIT vi sia un numero minore di contatti, generalmente hanno una durata maggiore. Anche la durata dei contatti influenza la capacità di scambio di messaggi all'interno della MSN: in Infocom5, due nodi che entrano in contatto hanno, generalmente, poco tempo (in media un minuto) per eseguire

la fase di discovery dei dispositivi, scambiarsi le informazioni necessarie a SIDEMAN$_{PQ}$ per valutare la possibilità di invare ad un nodo un messaggio, ed eseguire la fase di trasferimento dei messaggi.

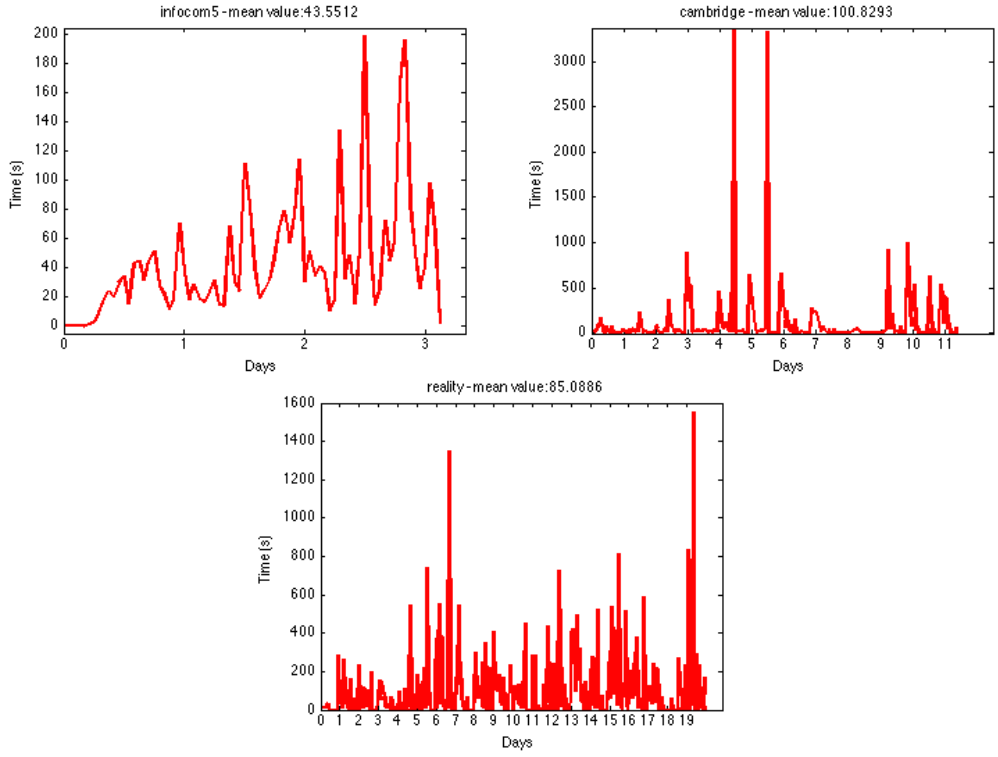


Figura 4.4: Durata media dei contatti

Un'altro aspetto che caratterizza le tracce di mobilità è il confronto tra il **numero di incontri totali ed il numero di nodi coinvolti**. I grafici in Figura 4.5 mostrano quanti utenti sono stati effettivamente incontrati (*Unique contacts*) rispetto al totale dei contatti (*Encounter*). Tale grafico è indice del fatto che i nodi tendono a frequentare un sottoinsieme della popolazione totale della simulazione; infatti:

- in Infocom5 un nodo frequenta il 93% della popolazione;
- in Cambridge un nodo frequenta l'83% della popolazione;
- in Reality MIT un nodo frequenta il 34% della popolazione.

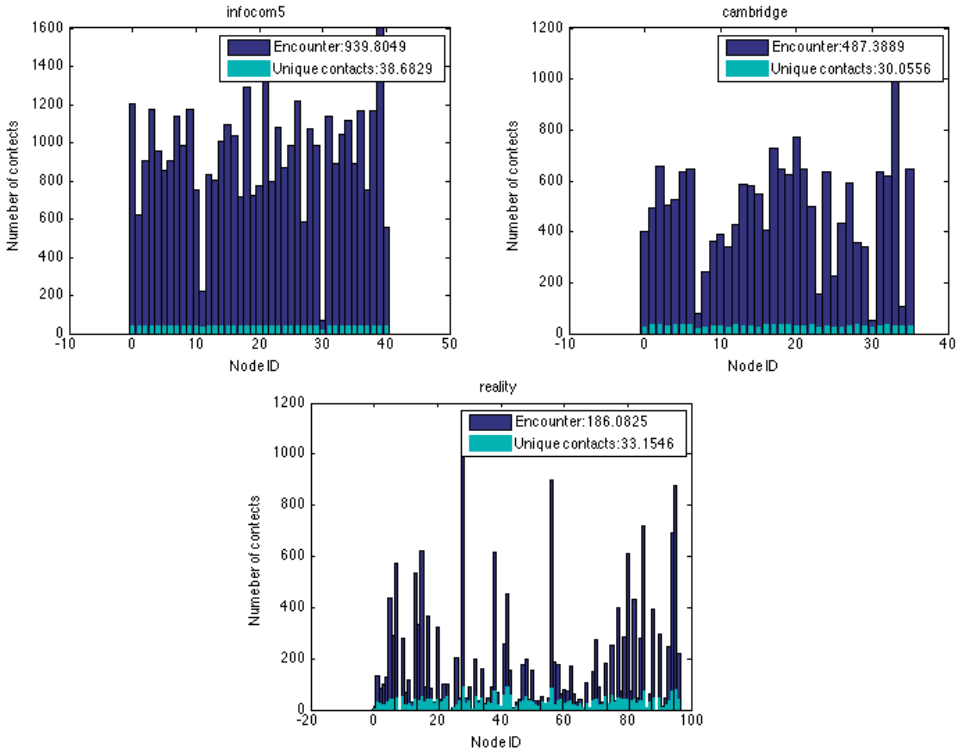


Figura 4.5: Numero di incontri e numero di incontri unici

L'ultimo aspetto temporale studiato è la **distribuzione dei tempi di intercontatto** tra i nodi (Figura 4.6). Come mostrato anche in [LOLG09], i tempi di intercontatto per questo tipo di tracce seguono uno specifico andamento:

- fino ad un certo valore, circa 10^3 in Infocom5 e circa 10^4 in Cambridge e Reality MIT, segue l'andamento di una power law,
- dopodiché ha un andamento di decrescita esponenziale.

Questo andamento ha ricevuto la seguente motivazione:

- gli incontri con minor tempo di intercontatto sono definiti in base ad una distribuzione di Zipf, che segue quindi la power law,
- invece gli incontri con maggiore tempo di intercontatto, definiti come "incontri futuri", seguono una distribuzione esponenziale.

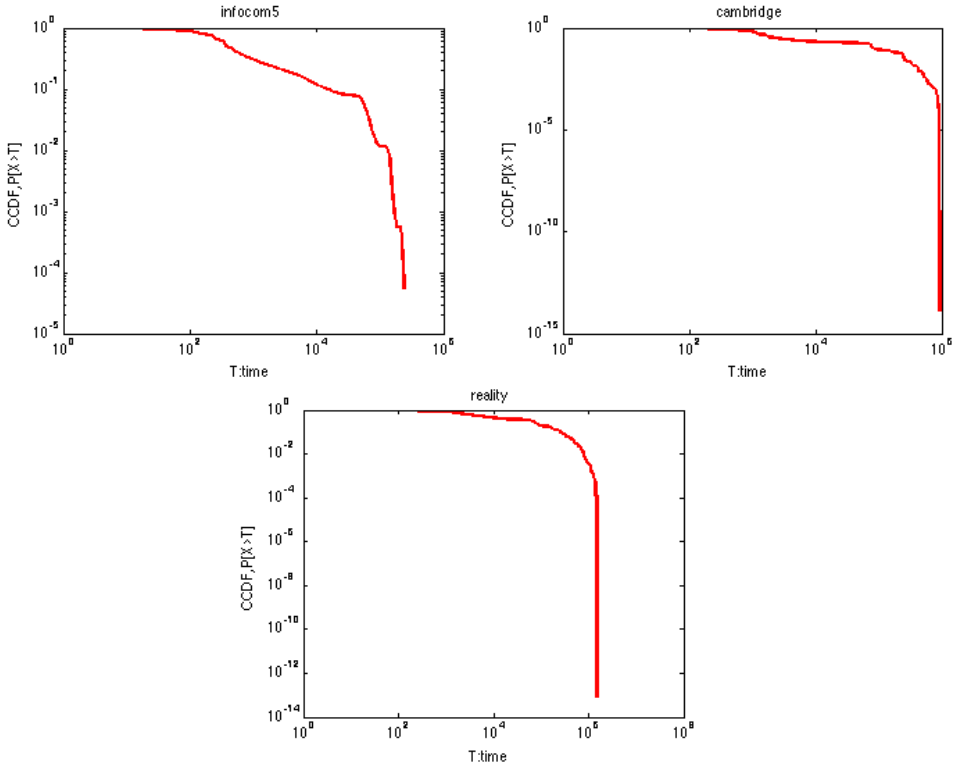


Figura 4.6: Distribuzione complementare dei tempi di intercontatto (CCDF)

4.2 Ambiente di simulazione

Durante la fase di test, l'algoritmo è stato eseguito in differenti scenari con differenti caratteristiche, ad esempio densità dello scenario, profilo dei nodi, valocità di spostamento dei nodi, etc. Questi test hanno permesso di studiare la variazione delle performance di SIDEMAN$_{PQ}$ in diversi scenari.

Gli scenari di simulazione richiedono di configurare l'ambiente in modo da riprodurre il comportamento umano. In particolare:

- le tracce di mobilità, che descrivono i contatti tra i nodi;
- il profilo dei nodi, come il numero di interessi associati ad ogni nodo;
- le caratteristiche della popolazione (il numero di nodi presenti nella simulazione, la distribuzione con la quale viene effettuata una query e quella con la quale un nuovo service advertisement è memorizzato nella cache di un nodo, etc);
- le caratteristiche di SIDEMAN$_{PQ}$ (il TTL iniziale associato alle query, etc).

Parleremo di queste ultime nel paragrafo 4.2.2.3, quando presenteremo l'implementazione di SIDEMAN$_{PQ}$ nel simulatore ONE; in questa sezione affrontiamo gli altri aspetti sopra elencati.

4.2.1 Modellazione del comportamento sociale

Un **nodo** in una MSN è profilato attraverso un certo *numero di interessi*. Un interesse è un argomento che può atttrarre uno o più nodi, ad esempio, il calcio o il cinema. L'insieme degli interessi di un nodo determina il suo comportamento durante l'esecuzione dell'algoritmo:

- durante la fase proattiva, un nodo scambierà advertisement con altri nodi i cui interessi sono simili a quelli di tale advertisement;
- durante la fase reattiva, un nodo invierà una query a nodi che hanno interessi simili a quelli di tale query.

All'interno di ogni scenario è possibile specificare il valore della dimensione dell'insieme degli interessi dei nodi, in base alle necessità della simulazione. Dato l'insieme $\mathcal{I}$ degli interessi di tutti i nodi nella rete, devono essere estratti ed assegnati dei sottoinsiemi di $\mathcal{I}$ ai vari nodi. Questa fase è stata sviluppata associando ad ogni nodo un numero stabilito e costante di interessi per tutta la durata della simulazione: si tratta di un'approssimazione della realtà utilizzata nei test per evitare l'implementazione della variabilità degli interessi di un nodo. Tuttavia, consideriamo tale scelta ammissibile, poiché riteniamo che gli interessi varino più lentamente rispetto alla durata degli scenari [BCP08a]. Dato un nodo n_j, i suoi interessi sono selezionati da $\mathcal{I}$ con un meccanismo di selezione che utilizza una distribuzione di Zipf, con parametro s, per definire quanto è concentrata la distribuzione. Più s si avvicina ad 1 (la distribuzione si dice "skew"), più la distribuzione è concentrata, ed estrae topic di maggior interesse; più s tende a 0, più sparsa sarà la distribuzione dei topic. Poiché gli interessi delle persone tendono generalmente ad essere piuttosto simili, sono stati utilizzati valori di s tra 0.7 ed 1. La distribuzione di Zipf permette di modellare un comportamento comune; molti nodi, infatti, sono attratti da pochi interessi comuni, mentre pochi nodi sono attratti da interessi meno comuni.

L'ambiente di simulazione permette di configurare il comportamento sociale della **popolazione**. Di seguito riportiamo gli aspetti che riteniamo caratterizzino la socialità delle persone:

- il *numero di nodi* che compongono la rete: generalmente, nei test eseguiti, utilizzeremo il numero di nodi presenti nelle tracce di mobilità.

Tuttavia è possibile creare degli scenari personalizzati[1] in cui inserire un numero a scelta di nodi.

- il *numero di servizi* offerti dai nodi della rete: valore generalmente impostato al doppio dei nodi presenti nella rete, per garantire la presenza di almeno un servizio per ogni interesse. Infatti ogni service provider *sp* a cui è assegnato un servizio *s* deve contenere i topic ad esso associati tra quelli di suo interesse

$$\mathcal{I}_{sp} \cap \mathcal{I}_s \neq \varnothing$$

- il *numero di topic utilizzati per una query o per un advertisement*: per questioni di semplicità, ma senza perdere in generalità, è stato preferito utilizzare query e advertisement che avessero soltanto un topic associato. In questo modo è stata semplificata la fase di confronto tra gli insiemi degli interessi.

- la *distribuzione di generazione di query*: questo valore determina il numero di query generate dai nodi. La generazione delle query da parte dei nodi della rete è stata modellata attraverso un processo di Poisson di intensità λ. Nelle simulazioni è stato impostato $\lambda = 2$: questo significa che, con una certa periodicità, sono selezionati in media due nodi, scelti in maniera uniforme all'interno della popolazione. Tali nodi hanno il compito di generare query.

- la *distribuzione di generazione di advertisement*: questo valore determina la frequenza di generazione dei service advertisement, ovvero il numero di advertisement che i nodi memorizzano nella propria service cache senza interagire con altri nodi. Questa distribuzione permette di modellare il processo con il quale un nodo apprende informazioni su un servizio. Anche in questo caso è stato utilizzato un processo di Poisson di intensità $\mu = 1$.

Tali parametri sono stati impostati in accordo ad alcuni risultati noti in letteratura [BCP08a, BCP08b, LW11].

4.2.2 The ONE simulator

Il simulatore ONE (Opportunistic Networking Environment) [KOK09] è un ambiente di sviluppo ideato per simulare l'esecuzione di protocolli ed applicazioni in reti mobili ad-hoc ed in DTNs, ed altri scenari in cui non è possibile applicare i protocolli di comunicazione delle reti tradizionali. ONE permette di:

[1] Scenari descritti attraverso dei pattern di mobilità noti, come il Working Day Movement Model [EKKO08], messi a disposizione dal simulatore: utilizzando questi scenari è possibile configurare il numero di nodi, e molti altri aspetti, rendendo non necessario utilizzare una traccia di mobilità reale.

- generare i movimenti dei nodi basandosi su diversi modelli di mobilità, o utilizzando tracce di mobilità reali,
- scambiare messaggi tra i nodi utilizzando diverse strategie di routing messe a disposizione dal framework,
- visualizzare messaggi e movimenti degli utenti in real-time tramite un'interfaccia grafica.

ONE è stato appositamente realizzato per la simulazione di comunicazioni opportunistiche su DTNs, in modo da permettere lo sviluppo e l'analisi di protocolli in questi ambienti. ONE mette a diposizione anche strumenti per la creazione di scenari sintetici basati su modelli di mobilità e permette di utilizzare tracce di mobilità reale; inoltre offre un insieme di tool per la visualizzazione (GUI) dell'andamento della simulazione e per l'elaborazione dei dati raccolti attraverso un meccanismo di logging.

4.2.2.1 Panoramica

Le funzionalità più importanti offerte dal simulatore (Figura 4.7) sono la modellazione del *movimento dei nodi*, l'utilizzo di diverse *interfacce* per la *comunicazione tra i nodi*, gli algoritmi di *routing*, la *gesitione dei messaggi* in ingresso ed in uscita e la possibilità di eseguire *applicazioni* sui nodi della rete.

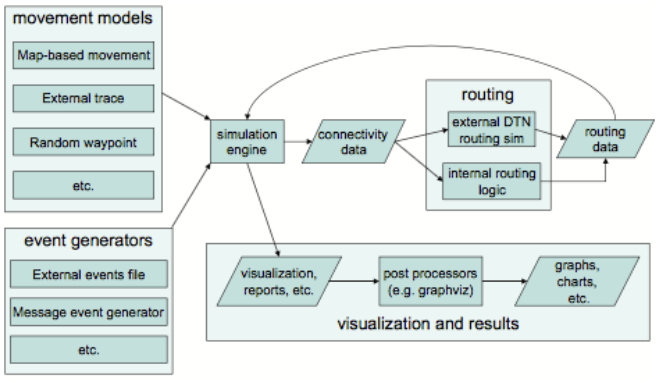


Figura 4.7: Panoramica di ONE

Possono essere utilizzate due tipologie di movimenti: quelli sintetici, attraverso l'uso di modelli di mobilità, e quelli reali, attraverso l'uso di tracce di mobilità reali. È possibile rilevare le connessioni opportunistiche tra in nodi in due modi diversi:

1. nel caso siano utilizzati movimenti sintetici, sono sfruttate la posizione dei nodi, il range trasmissivo ed il bit-rate dell'interfaccia di comunica-

zione utilizzata per stabilire l'esistenza o meno di un contatto tra due nodi;

2. nel caso siano utilizzati movimenti reali, sono sfruttati gli incontri descritti nella traccia di mobilità; ONE interpreta tracce di collocazione che utilizzano un formato che descrive gli incontri tra i nodi, indicando il tempo in cui è stato riscontrato un evento di connessione o disconnessione tra due nodi.

Per quanto riguarda le strategie di routing è possibile utilizzare quelle presenti nel framework, tra cui PRoPHET, Spray and Wait ed Epidemic, oppure crearne delle proprie, estendendo il simulatore.

4.2.2.2 La mobilità dei nodi in ONE

La mobilità dei nodi presenti nella simulazione è svolta utilizzando le tracce di mobilità presentate nel paragrafo 4.1. ONE utilizza delle tracce di collocazione con il formato mostrato in Figura 4.8.

time	CONN	node 1	node 2	up/down

Figura 4.8: Formato tracce di mobilità

Analiziamo il formato della traccia; essa è composta da diversi campi:

- il campo *time* identifica il tempo della simulazione, in secondi, in cui si è verificato tale evento;
- il campo *CONN* indica il tipo di evento, in questo caso un evento di connessione tra due utenti (CONN);
- i campi *node 1* e *node 2* identificano i nodi coinvolti nell'evento;
- l'ultimo campo identifica se è stata stabilita (up) o si è conclusa (down) la connessione.

Con questo tipo di tracce si definiscono le connessioni tra i nodi presenti nella rete: due nodi possono comunicare nell'intervallo di tempo tra il tempo corrispondente al recordi di connection up, e quello di connection down.

Un'altra tipologia di tracce utilizzate da ONE sono quelle che sfruttano la posizione dei nodi all'interno di un'ambiente. In questo tipo di tracce la connettività tra i nodi è ricavata dalle interfacce di cui sono forniti i nodi: infatti è possibile specificare il range trasmissivo ed il bit rate di tali interfacce per determinare gli incontri tra i nodi. In questo progetto di tesi non sono state utilizzate questo tipo di tracce, dato che lo studio si è concentrato su scenari reali.

4.2.2.3 Implementazione di SIDEMAN$_{PQ}$

In ONE un'applicazione può essere sviluppata utilizzando l'application layer del framework. Un modulo applicativo permette di: *(i)* ricevere messaggi, *(ii)* modificare messaggi alterando, aggiungendo o rimuovendo valori o contenuti, *(iii)* segnalare al layer di routing di inviare un messaggio, *(iv)* generare messaggi e *(v)* eseguire operazioni ad ogni step temporale della simulazione. Un modulo applicativo è associato ai nodi specificati nello scenario: per la sperimentazione di SIDEMAN$_{PQ}$ a tutti i nodi della rete è stato assegnato il modulo implementato per eseguire l'algoritmo.

Per sviluppare SIDEMAN$_{PQ}$ è stato esteso l'application layer di ONE: in particolare sono stati implementati il metodo per la gestione dei messaggi in ingresso (`handle`), e quello per l'aggiornamento dello stato del nodo (`update`), seguendo le regole e gli algoritmi definiti nel paragrafo 3.2. In particolare, nel metodo `update` sono eseguite la fase reattiva e la fase proattiva.

L'implementazione di SIDEMAN$_{PQ}$ ha un certo numero di parametri, utilizzati per variare il comportamento dell'algoritmo a seconda dello scenario analizzato:

- il valore massimo del TTL di una query: un nodo, prima di inoltrare una query decrementa e controlla questo valore, in modo da evitare l'inoltro del messaggio in caso sia minore o uguale a 0; questo parametro è utilizzato per evitare un'eccessiva diffusione di query nella rete.
- la frequenza con la quale eseguire la fase proattiva di SIDEMAN$_{PQ}$: come abbiamo visto in Figura 3.8, la fase proattiva dell'algoritmo è eseguita se uno o più nodi sono aggiunti al neighborhood, oppure se scade un timer; questo valore corrisponde a tale timer, ed è stato impostato a 3600 secondi; se nell'arco di tempo di un orail neighborhood non cambia, allora viene eseguita la fase proattiva.
- la soglia minima τ utilizzata nel calcolo della similarità: questo valore è utilizzato come soglia minima del valore di similarità, calcolo generalmente utilizzato sugli insiemi degli interessi.

4.3 Simulazioni

In questa sezione mostreremo i risultati preliminari ottenuti nella fase di sperimentazione dell'algoritmo SIDEMAN$_{PQ}$.

4.3.1 Metriche per service discovery

Per la valutazione di SIDEMAN$_{PQ}$ sono state utilizzate delle metriche il cui obiettivo è quello di mostrare le performance di un algoritmo di service discovery su MSNs. In particolare siamo interessati a valutare la precisione

degli advertisement scambiati tra i vari nodi in termini di interesse per i nodi (accuracy), ed anche, quante volte un nodo ha dovuto inviare una query per accedere ad un servizio, piuttosto che accedervi direttamente (proactivity). Mostreremo anche altre due metriche utilizzate generalmente nel service discovery: il numero medio di advertisement in cache, ed il tempo medio tra l'invio di una query e la ricezione della risposta.

Accuracy L'*accuracy* è una metrica che misura la precisione dell'algoritmo di service discovery nella propagazione degli advertisement ai nodi interessati ai servizi pubblicizzati. È definita come il rapporto tra il numero totale di advertisement presenti nella service cache di interesse per tale nodo, ed il numero totale di advertisement. Ovviamente si tratta di un valore compreso nell'intervallo $[0, 1]$.

Proactivity La *proactivity* è definita come il rapporto tra in numero di volte in cui un advertisement richiesto è trovato nella service cache, ed il numero totale di volte in cui un advertisement è cercato nella service cache. Questa metrica misura la qualità della fase dell'algoritmo di service discovery in cui sono inviati degli advertisements a nodi che potrebbero esservi interessati. Questo valore è compreso in un intervallo tra $[0, 1]$.

Caching Il *caching* è una metrica che misura la dimensione media della service cache dei nodi. Questa metrica mostra l'efficacia della fase di scambio degli advertisement.

Query response time Il *query response time* è definito come il tempo medio trascorso tra l'invio di una query e la ricezione del primo advertisement che risponde a tale query. Questa metrica misura l'efficacia del protocollo di service discovery nella fase di ricerca di advertisement.

4.3.2 Analisi dei risultati

Nella parte finale di questa tesi è stata eseguita una fase preliminare di sperimentazione dei SIDEMAN$_{PQ}$. Mostreremo le performance dell'algoritmo secondo le metriche appena definite. Nei grafici che seguono, il valore dell'*accuracy* rimane costante ad 1 per tutta la durata della simulazione: questo perchè l'algoritmo, come visto nel paragrafo 3.2, invia advertisement soltanto ai nodi che hanno una corrispondenza per gli interessi dell'advertisement.

Infocom5 I risultati di SIDEMAN$_{PQ}$ sulla traccia di mobilità Infocom5 sono riportati in Figura 4.9. L'algoritmo in questo scenario ha ottenuto dei risultati intermedi. Il valore dell'accuracy è rimasto costante, quindi gli

advertisement ricevuti dai nodi erano coerenti con i propri interessi. Il valore della proactivity ha raggiunto un valore superiore di 0.8 a fine simulazione, ed anche il valore del caching ha avuto un aumento costante.

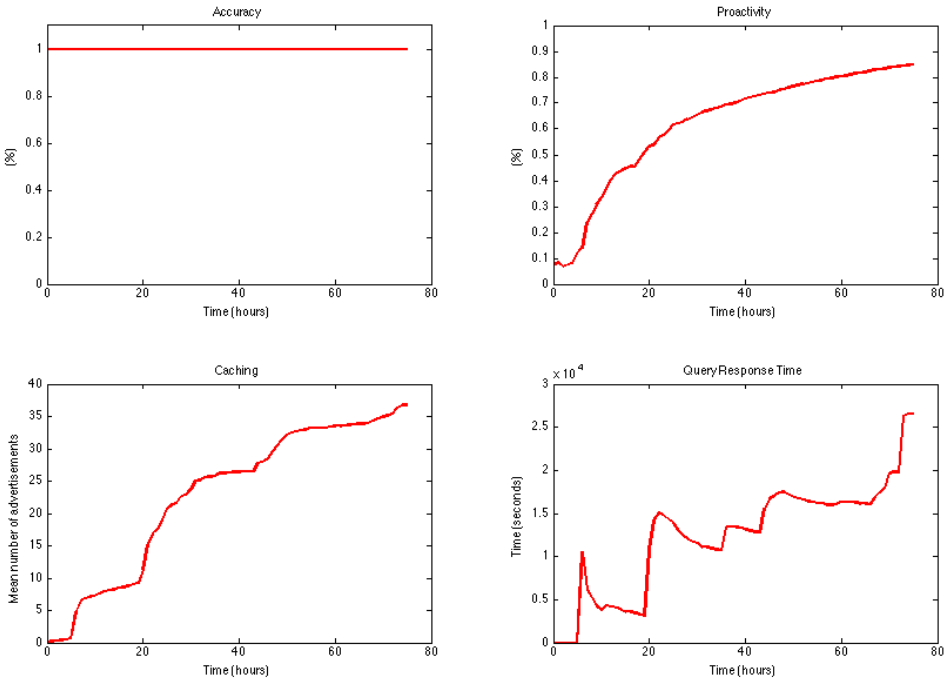


Figura 4.9: Risultati SIDEMAN$_{PQ}$ su Infocom5

Il numero di servizi durante questa simulazione è stato impostato ad 80, il doppio del numero dei nodi: come vediamo la dimensione finale della cache è di 35-40 advertisement. Come possiamo vedere il Figura 4.9, il valore della proactivity e quello del caching sono correlati: nell'intervallo temporale $[0, 40]$ abbiamo una rapida crescita di entrambi i valori, mentre nel resto della simulazione entrambi i valori crescono più lentamente.

Pur non raggiungendo un valore costante, sinonimo di fine della fase di querying, il query response time ha dei valori accettabili per questa fase preliminare di sperimentazione, anche se necessita di essere maggiormente investigato.

Cambridge I risultati di SIDEMAN$_{PQ}$ sulla traccia di mobilità Cambridge sono riportati in Figura 4.10. Questa traccia di mobilità ha riscontrato le migliori performance dell'algoritmo. Il valore dell'accuracy anche in questo caso è rimasto costante, quindi i nodi hanno esclusivamente advertisement di interesse nella propria service cache. Il valore della proactivity a fine simulazione ha raggiunto un valore superiore a 0.9, e le dimensioni della cache, dopo 150 ore dall'inizio della simulazione, si stabilizzano ad un valore che

è circa la metà dei servizi presenti nella rete; questo significa che, dopo un certo istante temporale, i nodi hanno tutti gli advertisement di cui hanno bisogno in cache.

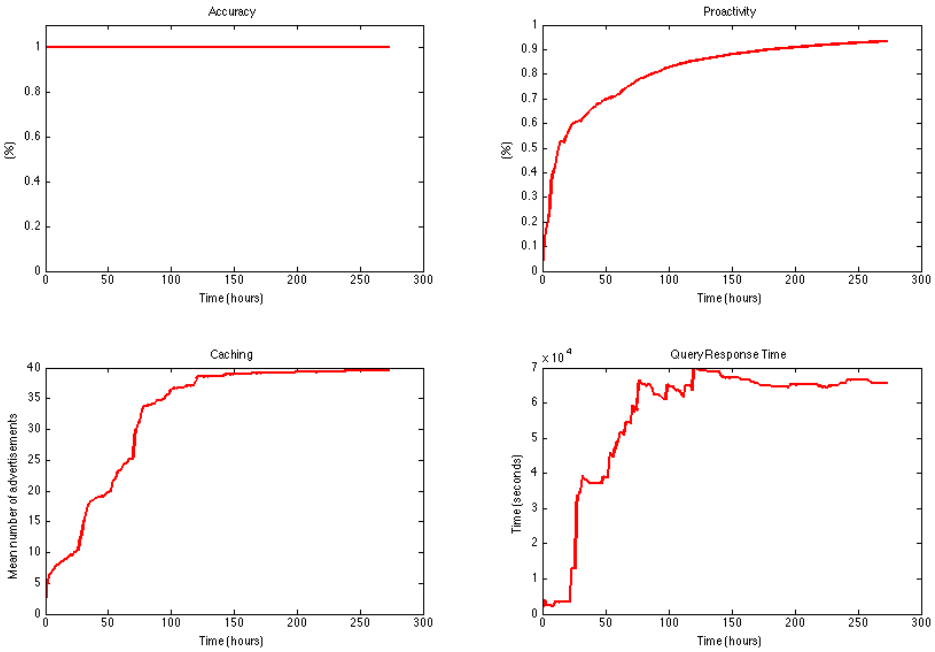


Figura 4.10: Risultati SIDEMAN$_{PQ}$ su Cambridge

Anche il query response time si è stabilizzato nell'intervallo di tempo $[150, 300]$: questo è un segnale, insieme alla stabilità della dimensione della cache, che tutti gli advertisement che potevano essere cercati sono stati trovati, e, quindi, le query risposte, per cui non è più necessario inoltrare richieste. In 4.1.2, Cambridge è stata definita una traccia di mobilità con caratteristiche intermedie tra Infocom5 e Reality MIT: probabilmente le caratteristiche riscontrate in questa traccia si adattano meglio all'esecuzione di un protocollo di service discovery.

Reality MIT I risultati di SIDEMAN$_{PQ}$ sulla traccia di mobilità Reality MIT sono riportati in Figura 4.11. SIDEMAN$_{PQ}$ per questa traccia di mobilità non ha avuto dei risultati molto soddisfacenti. Il valore dell'accuracy, come nei casi precedenti, è rimasto costante, quindi i nodi hanno nelle proprie service cache esclusivamente advertisement di interese. Il valore della proactivity è rimasto inferiore allo 0.8, e questo significa che l'algoritmo, anche a fine simulazione, continua a richiedere l'invio di query per ricevere informazioni sui servizi. Questo aspetto si nota anche sul valore del caching: infatti, come vediamo in Figura 4.11, è presente una corrispondenza tra la

crescita della proactivity ed il numero di advertisement in cache. Il numero di servizi impostato per la simulazione su Reality MIT è di circa 200, mentre il numero di advertisement presenti in cache a fine simulazione è soltanto di 45; questo significa che neanche la metà dei servizi è riuscita a circolare durante la simulazione. Un altro aspetto particolare dei risultati ottenuti nella traccia è forte rialzo del query response time avuto dopo metà simulazione.

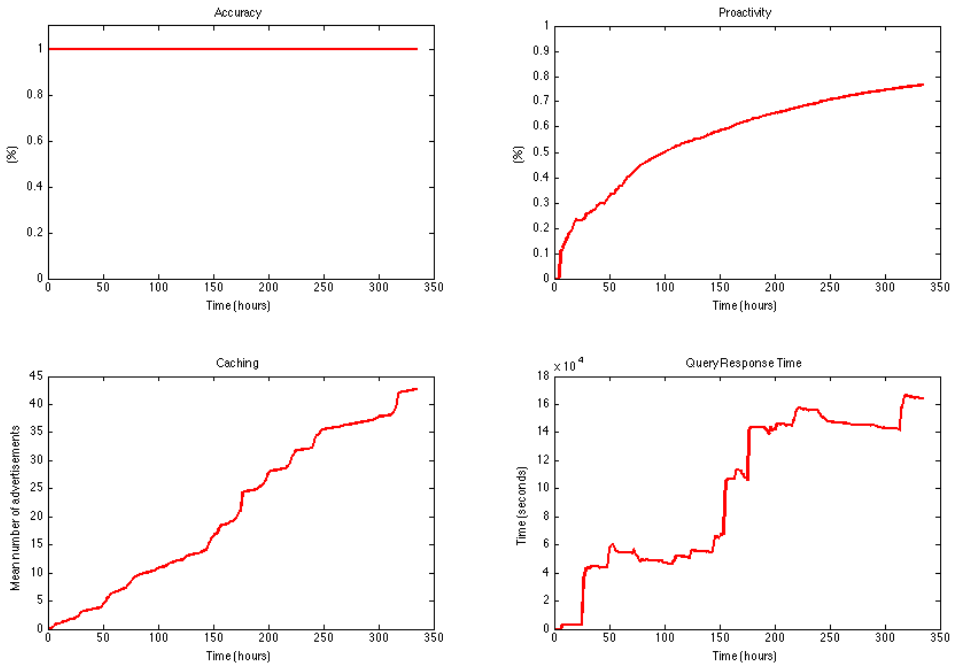


Figura 4.11: Risultati SIDEMAN$_{PQ}$ su Reality MIT

Molti aspetti di queste simulazioni hanno la necessità di essere maggiormente investigate, data la forte differenza tra il comportamento dell'algoritmo per le diverse tracce di mobilità.

Capitolo 5

Conclusioni e sviluppi futuri

In questa tesi è stato presentato SIDEMAN$_{PQ}$, un algoritmo di service discovery per reti mobili e sociali. Questa soluzione ha cercato di ottimizzare la precedente soluzione, SIDEMAN, in modo da migliorare le prestazioni delle fasi di service discovery e di service dissemination.

L'algoritmo è stato realizzato per l'applicazione a reti mobili e sociali, le MSN. Questo tipo di reti ha due caratteristiche principali: la mobilità umana ed i legami sociali. La mobilità umana, generalmente, è caratterizzata da tre aspetti principali: *(i)* le attività comuni tra più individui, *(ii)* il limitato numero di luoghi visitati, e *(iii)* la tendenza a seguire i percorsi più brevi negli spostamenti da un luogo ad un altro. I legami sociali, invece, differenziano i rapporti presenti tra i vari nodi: due nodi, ad esempio, possono essere amici, colleghi, parenti o sconosciuti. Abbiamo visto che queste relazioni influenzano gli incontri dei nodi all'interno di queste reti: generalmente due amici, o due colleghi, si vedono per una arco di tempo diverso, e con una frequenza diversa. Le caratteristiche temporali di tali relazioni possono fornire un meccanismo di classificazione delle relazioni sociali, e dare la possibilità di identificare gruppi di utenti con particolari caratteristiche, utili in fase di inoltro dei messagi. L'algoritmo sviluppato considera entrambi gli aspetti caratteristici delle mobile social networks:

- la mobilità dei nodi è sfruttata per la diffusione di informazioni all'interno della rete;
- gli aspetti sociali sono utilizzati per selezionare le informazioni di iteresse per i destinatari.

L'algoritmo SIDEMAN$_{PQ}$ è stato sviluppato per essere eseguito su architetture distribuite per il service discovery di tipo directory-less.

L'algoritmo sfrutta due tipologie di inoltro conosciute nell'ambito del problema del service discovery: l'inoltro proattivo e l'inoltro reattivo.

Nella fase reattiva, i nodi richiedono un certo servizio: se possono accedervi utilizzando un advertisement presente nella service cache, allora inoltrano una richiesta al service provider di tale servizio, altrimenti eseguono una query nella rete, in cerca di un advertisement per tale servizio. Nella fase proattiva, i nodi eseguono due fasi:

- una fase di scambio di informazioni sui servizi, nella quale inoltrano gli advertisement presenti nelle proprie service cache, utilzzando regole di selezione che considerano gli aspetti sociali di cui abbiamo parlato.
- una fase di scambio delle query che non hanno ancora ricevuto risposta: i nodi utilizzano infatti la struttura *pending query* per memorizzare tutte le query a cui non hanno ancora ricevuto risposta.

Un'aspetto molto importante considerato in SIDEMAN$_{PQ}$ è quello delle comunità: il fatto che il nodo n_i si trovi nella comunità del nodo n_k indica la presenza di una relazione di qualche tipo tra questi due nodi. Questa struttura, ottenuta utilizzando algoritmi di community detection, è stata sfruttata in SIDEMAN$_{PQ}$ nella fase di inoltro dei messaggi: infatti, il legame presente all'interno delle comunità indica un certo livello di similarità tra i nodi, e questo indica, quindi, la possibilità di scambiare advertisement di interesse.

Il processo che ha portato allo sviluppo di SIDEMAN$_{PQ}$, quindi, ha visto inizialmente una fase di studio dello scenario delle MSNs e di analisi delle tracce di mobilità: in questo modo è stato possibile ricavare alcune delle proprietà utilizzate dall'algoritmo. Successivamente è stato analizzato l'algoritmo esistente, e sono state ideate le strategie per la realizzazione di un protocollo migliore. Infine è stato sviluppato e testato l'algorimo sul simulatore ONE.

La fase di sperimentazione presenta possibilità di ulteriore sviluppo, estendendola ad ulteriori dataset oltre a quelli considerati nella tesi.

Un'altro aspetto interessante da analizzare potrebbe essere quello della ciclicità delle tracce di mobilità: durante la tesi, infatti, è stata individuata l'importanza della struttura della mobilità nel caso dell'inoltro dei messaggi. Potrebbe essere interessante riuscire a definire in qualche modo questo aspetto di ciclicità, e studiare le implicazioni che ha sull'invio di query ed advertisement all'interno di SIDEMAN$_{PQ}$.

Infine, un ulteriore aspetto da analizzare, potrebbe essere lo studio di tecniche di offloading, per spostare parte del calcolo dai nodi ad una struttura cloud-based. L'utilizzo di questo tecniche potrebbe alleggerire il compito dei nodi: ad esempio, la fase di community detection, dove sono scambiati un certo numero di informazioni, potrebbe essere eseguita in maniera centralizzata, piuttosto che localmente da parte dei nodi; infatti un servizio cloud

potrebbe implementare un algoritmo di community detection centralizzato, più precisi e rapidi di quelli distribuiti.

Bibliografia

[Apa98] ApacheJiniSpecifications. Jini Architecture Specification. 1998.

[App04] Apple. Apple Support, 2004.

[Bar69] John A Barnes. Graph theory and social networks: A technical comment on connectedness and connectivity. *Sociology*, 3(2):215–232, 1969.

[BCP08a] Chiara Boldrini, Marco Conti, and Andrea Passarella. Contentplace: Social-aware data dissemination in opportunistic networks. In *Proceedings of the 11th International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '08, pages 203–210, New York, NY, USA, 2008. ACM.

[BCP08b] Chiara Boldrini, Marco Conti, and Andrea Passarella. Context and resource awareness in opportunistic network data dissemination. In *World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a*, pages 1–6. IEEE, 2008.

[BCP11] Eleonora Borgia, Marco Conti, and Andrea Passarella. Autonomic detection of dynamic social communities in Opportunistic Networks. pages 142–149, 2011.

[BMD13] Paolo Bellavista, Rebecca Montanari, and Sajal K Das. Mobile social networking middleware: A survey. *Pervasive and Mobile Computing*, 9(4):437–453, 2013.

[BP10] Chiara Boldrini and Andrea Passarella. Hcmm: Modelling spatial and temporal properties of human mobility driven by user social relationships. *Computer Communications*, 33(9):1056–1074, 2010.

[CJYF06] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, and Tim Finin. Toward distributed service discovery in pervasive computing environments. *Mobile Computing, IEEE Transactions on*, 5(2):97–112, 2006.

[CMMP08] P. Costa, C. Mascolo, M. Musolesi, and G.P. Picco. Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks. *Selected Areas in Communications, IEEE Journal on*, 26(5):748–760, June 2008.

[CMP+05] Celeste Campo, Mario Munoz, Jose Carlos Perea, A Mann, and Carlos Garcia-Rubio. Pdp and gsdl: a new service discovery middleware to support spontaneous interactions in pervasive systems. In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pages 178–182. IEEE, 2005.

[EKKO08] Frans Ekman, Ari Keränen, Jouni Karvo, and Jörg Ott. Working day movement model. In *Proceedings of the 1st ACM SIGMOBILE workshop on Mobility models*, pages 33–40. ACM, 2008.

[EP05] Nathan Eagle and Alex (Sandy) Pentland. CRAWDAD data set mit/reality (v. 2005-07-01). Downloaded from http://crawdad.org/mit/reality/, July 2005.

[FPG12] Alan Ferrari, Daniele Puccinelli, and Silvia Giordano. Characterization of the impact of resource availability on opportunistic computing. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 35–40. ACM, 2012.

[GBCF13] Michele Girolami, Paolo Barsocchi, Stefano Chessa, and Francesco Furfari. A social-based service discovery protocol for mobile ad hoc networks. In *Ad Hoc Networking Workshop (MED-HOC-NET), 2013 12th Annual Mediterranean*, pages 103–110. IEEE, 2013.

[GBFC14] Michele Girolami, Stefano Basagni, Francesco Furfari, and Stefano Chessa. SIDEMAN : ServIce DiscovEry in Mobile sociAl Networks. 2014.

[GN02] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[Gra73] Mark S Granovetter. The strength of weak ties. *American journal of sociology*, pages 1360–1380, 1973.

[HCY11] Pan Hui, J. Crowcroft, and E. Yoneki. Bubble rap: Social-based forwarding in delay-tolerant networks. *Mobile Computing, IEEE Transactions on*, 10(11):1576–1589, Nov 2011.

[HDVL03] Sumi Helal, Nitin Desai, Varun Verma, and Choonhwa Lee. Konark-a service discovery and delivery protocol for ad-hoc networks. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 3, pages 2107–2113. IEEE, 2003.

[HHK+12] Bo Han, Pan Hui, V.S. Anil Kumar, Madhav V. Marathe, Jianhua Shao, and Aravind Srinivasan. Mobile Data Offloading through Opportunistic Communications and Social Participation. *IEEE Transactions on Mobile Computing*, 11(5):821–834, May 2012.

[Hum10] L. Humphreys. Mobile social networks and urban public space. *New Media & Society*, 12(5):763–778, February 2010.

[HYCC07] Pan Hui, Eiko Yoneki, Shu-yan Chan, and Jon Crowcroft. Distributed Community Detection in Delay Tolerant Networks. 2007.

[JX13] Behrouz Jedari and Feng Xia. A survey on routing and data dissemination in opportunistic mobile social networks. *CoRR*, abs/1311.0347, 2013.

[KKO10] Ari Keränen, Teemu Kärkkäinen, and Jörg Ott. Simulating Mobility and DTNs with the ONE (Invited Paper). *Journal of Communications*, 5(2):92–105, February 2010.

[KKRO03] M. Klein, B. Konig-Ries, and P. Obreiter. Service rings - a semantic overlay for service discovery in ad hoc networks. In *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pages 180–185, Sept 2003.

[KOK09] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. The ONE Simulator for DTN Protocol Evaluation. In *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, New York, NY, USA, 2009. ICST.

[KT04] Ulas C. Kozat and Leandros Tassiulas. Service discovery in mobile ad hoc networks: an overall perspective on architectural choices and network layer support issues . *Ad Hoc Networks*, 2(1):23 – 44, 2004.

[LCC12] K.C.-J. Lin, Chun-Wei Chen, and Cheng-Fu Chou. Preference-aware content dissemination in opportunistic mobile social networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 1960–1968, March 2012.

[LDS03] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(3):19–20, July 2003.

[LKM07] V. Lenders, G. Karlsson, and M. May. Wireless ad hoc podcasting. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON '07. 4th Annual IEEE Communications Society Conference on*, pages 273–283, June 2007.

[LOLG09] Uichin Lee, Soon Y Oh, Kang-Won Lee, and Mario Gerla. Scaling properties of delay tolerant networks with correlated motion patterns. In *Proceedings of the 4th ACM workshop on Challenged networks*, pages 19–26. ACM, 2009.

[LSM+11] Nicolas Le Sommer, Yves Mahéo, et al. Olfserv: an opportunistic and location-aware forwarding protocol for service delivery in disconnected manets. In *Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (Ubicomm 2011)*, pages 115–122, 2011.

[LSSM08] Nicolas Le Sommer, Romeo Said, and Yves Mahéo. A proxy-based model for service provision in opportunistic networks. In *Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing*, pages 7–12. ACM, 2008.

[LW11] Cong Liu and Jie Wu. Practical Routing in a Cyclic MobiSpace. *IEEE/ACM Transactions on Networking*, 19(2):369–382, April 2011.

[MLSM12] Ali Makke, Nicolas Le Sommer, and Yves Mahéo. Tao: A time-aware opportunistic routing protocol for service invocation in intermittently connected networks. In *ICWMC 2012, The Eighth International Conference on Wireless and Mobile Communications*, pages 118–123, 2012.

[MPEP09] Benjamin Molina, Salvatore F. Pileggi, Manuel Esteve, and Carlos E. Palau. A negotiation framework for content distribution in mobile transient networks. *Journal of Network and Computer Applications*, 32(5):1000–1011, September 2009.

[MSLC01] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, pages 415–444, 2001.

[NR07] Tuan Dung Nguyen and Siegfried Rouvrais. A socially inspired peer-to-peer resource discovery service for delay tolerant networks. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, pages 960–969. Springer, 2007.

[OF13] M. Orlinski and N. Filer. The rise and fall of spatio-temporal clusters in mobile ad hoc networks. *Ad Hoc Networks*, 11(5):1641–1654, July 2013.

[PDFV05] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

[PKO12] Mikko Pitkanen, Teemu Karkkainen, and Jörg Ott. Mobility and service discovery in opportunistic networks. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 204–210. IEEE, 2012.

[POL+09] Anna-Kaisa Pietiläinen, Earl Oliver, Jason LeBrun, George Varghese, and Christophe Diot. MobiClique. In *Proceedings of the 2nd ACM workshop on Online social networks - WOSN '09*, page 49, New York, New York, USA, August 2009. ACM Press.

[Sal12] A. Salvini. *Connettere. L'analisi di rete nel servizio sociale.* ETS, 2012.

[SGC+06] James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot, and Augustin Chaintreau. CRAWDAD data set cambridge/haggle (v. 2006-01-31). Downloaded from http://crawdad.org/cambridge/haggle/, January 2006.

[SM12] Nor Hashimah Sulaiman and Daud Mohamad. A jaccard-based similarity measure for soft sets. In *Humanities, Science and Engineering Research (SHUSER), 2012 IEEE Symposium on*, pages 659–663. IEEE, 2012.

[Sun00] SunMicrosystems. Service Location Protocol Administration Guide, 2000.

[TRL12] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *Communications Surveys & Tutorials, IEEE*, 14(1):131–155, 2012.

[UPn08] UPnPForum. UPnP Specifications, 2008.

[VB00] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, 2000.

[VP08] C.N. Ververidis and G.C. Polyzos. Service discovery for mobile ad hoc networks: a survey of issues and techniques.

Communications Surveys Tutorials, IEEE, 10(3):30–45, Third 2008.

[VY13] Nikolaos Vastardis and Kun Yang. Mobile Social Networks: Architectures, Social Properties, and Key Research Challenges. 15(3):1355–1371, 2013.