Ad Hoc & Sensor Wireless Networks, Vol. 34, pp. 1–39
Reprints available directly from the publisher
Photocopying permitted by license only

SIDEMAN: Service Discovery in Mobile Social Networks

MICHELE GIROLAMI[1,3,*], STEFANO BASAGNI[2], FRANCESCO FURFARI[1], STEFANO CHESSA[1,3]

[1]*Italian National Council of Research, Via G. Moruzzi 1 56124, Pisa*
E-mail: furfari@isti.cnr.it
[2]*ECE Department, Northeastern University, 312 Dana Research Center 360 Huntington Ave Boston, MA 02115 USA*
E-mail: basagni@ece.neu.edu
[3]*Department of Computer Science, Largo B. Pontecorvo, 3, 56127 Pisa, Italy*
E-mail: ste@di.unipi.it

Received: November 16, 2014. Accepted: September 21, 2015.

In this paper we present SIDEMAN, a service discovery algorithm that exploits human mobility patterns in Mobile Social Networks (MSN). SIDEMAN takes advantage of two aspects of MSN, namely, that users tend to form communities, and that users in the same community share interests for similar services. The performance of SIDEMAN has been evaluated through simulations in real and synthetic scenarios: A set of traces collected at IEEE Infocom 2006 and traces obtained from the HCMM mobility model, respectively. We have compared our algorithm to the social version of two popular discovery techniques, namely, flooding and gossiping. We investigated the following key metrics: How proactive an algorithm is in distributing services of interest (Recall); how many services are already with a user when s/he needs them (Gain); the energy cost necessary for service discovery; the time needed to reply to a service query, and the average number of services stored and exchanged. Our results show that in all considered scenarios SIDEMAN is remarkably effective in obtaining flawless Recall and a Gain that is always comparable to that of the other algorithms. Furthermore, most services are retrieved in reasonable time and at a remarkably lower energy cost than that of flooding and gossiping-based solutions.

Keywords: Service discovery, mobile social networks, distributed computing.

[*] Contact author: E-mail: michele.girolami@isti.cnr.it

1 INTRODUCTION

Social networking has being knowing increased notoriety thanks to the widespread use of mobile devices, especially smart phones and tablets, whose pervasive presence constitutes the vast majority of devices at the edge of modern telecommunication networks. A new paradigm of networking has emerged as a consequence, which capitalizes on the opportunistic nature of human encounters to implement fundamental network services such as content distribution [23], routing [21], and service discovery [29, 30]. This new paradigm is often called *mobile social networking* to emphasize the human interaction on which it is based [43], or *pocket switched networking*, to indicate the kind of devices of which it is made [37].

The key ideas of Mobile Social Networks (MSN) is that the end user can actively participate in content distribution instead of just being a recipient of information from the telecommunication infrastructure. For these reasons, studies on the mobility of humans and on their social relationships have flourished with the intent of investigating how people, through the increasingly capable devices in their pockets, become actors in content distribution. For instance, for opportunistic data forwarding it becomes important to know how humans move and whether there are common patterns in their movements [12, 45, 48]. Routing can take advantage of knowing that people tend to visit a restricted number of locations, or that they prefer short paths instead of longer ones [7, 13, 22].

In this paper we contribute to the problem of service discovery in MSN. Service discovery concerns determining the existence of services in a network by giving the service providers the possibility of announcing the existence of a service, and to those interested in a service the capability to find the services they need. As a problem for general networking, service discovery has been widely investigated [10]. Early solutions, including algorithms such as Jini [41], UPnP [33], SLP [18] and Bonjour [2], concern service discovery for infrastructure-based IP networks, and as such are not suitable for mobile networks such as MSN. More recently, schemes have been proposed for general mobile networks, such as ad hoc networks [17, 44], and for pervasive communications [9]. These solutions concern general wireless networks, describing centralized and distributed protocols and middleware that do not take into account the peculiar characteristics of MSN. In particular they do not consider the all-human tendency to keep visiting familiar places and to form communities composed by people with *similar* interests. The contribution of this work is a novel algorithm for service discovery in MSN. The algorithm, named SIDEMAN for *ServIce DiscovEry for Mobile sociAl Networks*, has been designed by taking into account some aspects characterizing how people move and how people interact with each other. In particular, SIDEMAN implements the two core operations of service discovery, namely:

- *Service dissemination*, i.e., the distribution of services to the MSN users. This operation is realized by discovering and recognizing social communities and by a proactive diffusion of services among people with similar interests.
- *Service query*, i.e., the process of requesting a needed service from a fellow user. This operation is implemented by a controlled query propagation mechanism aimed at avoiding extensive use of indiscriminate flooding.

All devices running SIDEMAN are peers without any specific role, and collaborate to disseminate services within their communities. Users proactively exchange services with one another for avoiding, as much as possible, that a query needs to be sent to obtain a needed service. One of the goals of SIDEMAN is to *anticipate* the user needs by exchanging, in advance, services that could be of interest to the user. If a user needs a service that is not currently available to it, s/he reactively crafts a corresponding query and sends it to the users currently in her/his own community.

In summary, the innovative aspects of SIDEMAN with respect to previous solutions are:

- The use of both reactive and proactive approaches to service discovery for actively submitting a query and for passively being notified with services of interests.
- The use of a community-based diffusion strategy for the propagation of queries and services. In particular, query and service messages are forwarded selectively to the members of a community whose interests match those of the message to be forwarded. Moreover, SIDEMAN keeps track of communities visited in the past so to avoid keeping detecting communities already known.

The performance of SIDEMAN has been evaluated via simulations in scenarios where users move according to the HCMM [7] mobility model and according to real traces collected at IEEE Infocom 2006 [19]. SIDEMAN has been compared to two other algorithms used to propagate services, called s-Flooding and s-Gossip. s-Flooding is a "social" version of common flooding [26], where instead of broadcasting services and queries to all users, the source sends them only to users in her/his own community. Similarly, in s-Gossip [25] each user spreads the services and queries only to a random number of fellow user in her/his own community. Selected performance metrics include the goodness of SIDEMAN in proactively provide services that a user might be interested in (Recall), its effectiveness in making available to the user just the services in which s/he is interested (Gain), the average energy consumed by a device for service discovery, the average time needed to receive a response to a service query and the dimension of the cache and the number of services exchanged. Our results show that SIDEMAN outperforms both s-Flooding and s-Gossip. In particular, in both synthetic (HCMM) and real-world (Infocom 06) scenarios, we observed that SIDEMAN achieves perfect Recall, being able to provide users with all and only the services they want, whereas s-Flooding and s-Gossip provide users with many uninteresting services. Despite the greatest number of services disseminated by s-Flooding and s-Gossip, which benefit the Gain of these two algorithms, SIDEMAN achieves a comparable Gain at a much lower cost in terms of overall energy consumption, number of services exchanged among nodes and number of services stored locally to each node. In other words, when a user running SIDEMAN needs a service, the probability of finding it in her/his own cache is equivalent to those of the other two algorithms. The best query response time is obtained by the algorithm that distributes the most services, namely, s-Flooding. However, SIDEMAN response time is only at most 12% higher than that of s-Flooding, and obtains this result by transmitting a considerably lower number of queries. (Both algorithms greatly outperform the query response time of s-Gossip.) Remarkably, all these SIDEMAN performance results are obtained at a fraction of the energy cost required by service discovery for users using s-Flooding and s-Gossip. In particular, devices running SIDEMAN are able to save up to 85.71% (84.61%) of their energy with respect to s-Flooding (s-Gossip).

The rest of the paper is organized as follows. Section 2 surveys previous works on algorithms and protocols for MSN. Section 3 introduces the reference scenario and provides a definition of the service discovery problem. In Section 4 we introduce notations and definitions used to describe the algorithm. Section 5 describes SIDEMAN in details. In Section 6 we evaluate the performance of SIDEMAN and compare it to that of s-Flooding and s-Gossip. Conclusions and future works are provided in Section 7.

2 RELATED WORKS

Traditional service discovery protocols have been investigated for over a decade [10, 17, 44]. Most of them are designed for networks with a static infrastructure or for mobile ad hoc networks. As mentioned, these protocols do not meet the typical requirements and constraints of MSN [4, 49].

Solutions to the problem of the service discovery in MSN can take advantage of a recent new and emerging class of protocols for data forwarding in social-based networks. In this section we illustrate the basic ideas behind these protocols and highlight those aspects of SIDEMAN that are inspired by those ideas and those who are instead a novel contribution.

Mei et al. introduce SANE, a social-aware, stateless algorithm for routing in Pocket Switched Networks [32]. According to SANE, movements of individuals are driven by interests. Information is therefore disseminated among users with similar interests (*interest-cast* diffusion model). When two users meet, they first exchange their interest profiles. As soon as a user has information to send to a destination, it forwards this information to the user whose interests match those of the destination. Similarly to SANE, SIDEMAN adopts an interest-based strategy. However, our algorithm relies on a community detection algorithm to select which nodes will receive both queries and services.

The authors of [36] defines the BehaviourCast problem for the diffusion of information in MSNs. BehaviourCast is based on four key-features. Validity concerns forwarding a message to a subset of the interested device. Effectiveness concerns forwarding the message so to achieve total coverage of interested devices. Efficiency concerns involving the smaller number of relying devices. Finally, termination has to do with interrupting the forwarding of a message after a given time. The authors also propose two interest-based strategies for the BehaviourCast problem, namely the basic InterestCast and the weighted InterestCast. With the basic InterestCast every device i executes a utility function that counts the total number of devices encountered that also share the same interests of i. The authors argue that such basic strategy has no memory of past encounters. For this reason the authors propose an enhanced version, namely, the weighted utility function. This last strategy, based on Shannon's entropy principle, counts separately the number of encounters that device i had with device j in the past. The weighted utility function, hence, keeps track of the number of encounters with every single devices met. The authors simulate the two strategies proposed both with real and synthetic mobility traces and they compare the results with those of two similar works, ProfileCast and SocialCast. The forwarding rule of SIDEMAN is similar to the (weighted) InterestCast algorithm. However SIDEMAN also implements a mechanism for recognizing communities already visited in the past, which reduces the community detection-imposed overhead.

The SocialCast [11] algorithm implements a publish/subscribe paradigm [5]. SocialCast relies on the observation that people with similar interests tend to meet more frequently than people without overlapping interests. The data forwarding strategy is implemented by observing the mobility patterns of people and also the interests of the devices running

SocialCast. SocialCast uses an interpolation function based on Kalman filters to predict the movement of people by tracking their previous movements. The SocialCast algorithms is implemented in steps: *(i)* Dissemination of user interests: asks to each device to broadcast the list of its interests to its 1-hop neighborhood. Then *(ii)* every device computes the utility function for all its interests, and *(iii)* every device checks the contents it carries with respect to the subscriptions of the devices, and eventually the contents are forwarded to the subscribers.

Nguyen and Rouvrais propose a socially-inspired resource discovery service for delay tolerant networks that exploits the Community-based Mobility Model (CMM) [34]. Resources and users are classified by interests, and the search is performed by first sending a query to neighbors with similar interests. If the resource is not found, the query is sent to every node within transmission range. SIDEMAN adopts a similar strategy for service query distribution. However, SIDEMAN also implements a proactive strategy for the diffusion of the services, aimed at providing users with services they might want to use. This strategy reduces the number of service queries, since once a user needs a service, s/he first check whether s/he has that service already, and sends a query only if that service is missing.

In [46] the authors address the problem of service discovery in delay tolerant networks. The scheme uses a proactive propagation of services, where each service provider announces periodically its own services to the entire network. The service is composed by a list of keywords, and all the devices in the network cache the received services and associate them to their latest time of reception. When a client looks for a service, it first checks into its local cache. If there are no matches, then it starts a reactive service discovery by broadcasting a service query message. Each device receiving the query looks for matches in its internal cache. As a consequence, the query can receive an early response even from intermediate devices. The client then selects matching replies to its query based on the latest time of reception. This solution is evaluated by means of NS-2-based simulations for assessing the efficiency and the overhead of service discovery. Differently from SIDEMAN, this scheme does not consider social aspects in the services and query distribution. We argue that the basic architecture of service discovery presented in [46] is also valid for MSN, and its mechanisms of diffusion of services and service queries can be easily combined with knowledge about the communities and user interests to be adopted in MSN.

OLFServ (Opportunistic and Location-aware Forwarding protocol for Service delivery) considers a scenario in which geo-localized devices that are deployed in a mobile ad hoc network [24]. The authors explicitly consider a sparse application scenario where the network may become disconnected for some time. It assumes that any device can advertise a service, by using a multicast-based scheme that limits the area of propagation of the service itself. Services have an expiration time and they contain information about the position of the emitter (the service provider) and the geographic area where the service can be accessed. Furthermore, services also include a list of potential recipients (service clients), although the paper does not discuss how this list is created and maintained. Beyond publicizing a new service, the purpose of the service is also to make more efficient the access to the service by providing geographical information about the service provider. The authors evaluate the efficiency of OLFServ in distributing the services against their rate of success (in terms of number of clients that find correctly a service provider).

The authors of [1] present a service discovery protocol for mobile ad hoc networks (MANET) with low mobility. The protocol is based on a reliable broadcast scheme. Services are not described by identifiers or by a service type. Rather, they are labeled with input/output parameters. The parameters of the services available in the network are spread using a proactive exchange of the local tables stored locally by each device. Such tables contain a mapping between the service and the I/O parameters needed. Moreover, the protocol is supposed to be embedded with the neighbors discovery protocol of the underlying MANET. In this way, the services are advertised together with the discovery of devices found in proximity. Queries are diffused reactively by flooding the neighborhood, i.e., as soon as a device needs to access to a specific service. The queries are described with a set of parameters. The use of parameters are described using a common taxonomy, that includes the possibility of hierarchies of parameters. The authors classify the queries within two categories, namely *exact* or *generic*. The protocol is evaluated via NS2-based simulations that investigate metrics such as the delay of service discovery and the amount of overhead generated by the protocol.

The authors of Delegation Forwarding [14] describe strategies for the diffusion of information in social-based networks. These strategies are all based on the idea of forwarding the information from the sender to the receiver only if the receiver is “better” than the source, where “better” is with respect to selected metrics. For instance, a receiver is deemed worth of receiving from the source, if the number of social contacts of the receiver is bigger than that of the contacts of the sender. SIDEMAN adopts a similar approach for the dissemination of services, allowing a sender to share a services only if the receiver is interested in it.

Socio-aware is a publish/subscribe strategy that relies on a overlay structure based on communities [50]. Communities are detected by means of two algorithms proposed by the same authors [20]. The overlay structure assumes that devices can play different roles: The brokers, the subscribers and the publishers. The brokers have high centrality degree inside the community. The brokers receive all the subscriptions and un-subscriptions from other devices as well as the list of the centrality values form the devices in contact with a time stamp. The broker device evaluates the centrality values previously received in order to decide which device should take the role of the broker. If a change is needed, then the broker transfers the subscription list to the new broker with the highest centrality degree. Then, an update message is sent to all the brokers in the network. During the gossiping stage, subscriptions are propagated towards the community broker. When a publication reaches the broker, it is propagated to all other brokers, and then each broker checks its own subscription list. In case there are members in its community that must receive the publication, the broker floods the community with the information.

The BUBBLE Rap algorithm implements a social-based forwarding strategy in delay tolerant networks [19]. Users are given a global ranking and a local ranking, computed according to their "importance" in the network. Information is first forwarded from the sender through users with higher global ranking, until it reaches a user in the same community of the receiver. It is then forwarded only among users in the same community according to the local ranking, until the information reaches its destination. SIDEMAN makes no use of hierarchical organization. Services and service queries are forwarded from the sender to the receiver only if the latter is interested in it. Furthermore, SIDEMAN implements a mechanism for recognizing communities already visited, which reduces the time and the resources spent for forwarding.

In proposing DIFFUSE, Lin et al. provide a solution for data dissemination in pocket switched networks [27]. The forwarding strategy is based on computing the *contribution* of each user, a parameter indicating the frequency and duration of contacts that a user has with other users. This parameter is used to select the relay to which data is forwarded. Both SANE and DIFFUSE make no use of the concept of community, thus saving in operations that have to do with community detection and recognition. SIDEMAN service dissemination and discovery instead makes use of community membership in that services and service queries are exchanged only between members of the same community that share the same interests. In order to reduce the overhead of community management, SIDEMAN implements an efficient mechanism for recognizing communities a user has visited in the past. This mechanism is shown to reduce the number of transmissions among users and therefore their device energy consumption.

Table 1 summaries the papers presented in this section according to three main categories: (1) Interest-based, (2) flooding-based, and (3) social-based strategies. The table also reports if the strategy described adopts a community

			Mobility Traces			Evaluation Metrics		
	Paper	Discovery Strategy	Real-world	Synthetic	Community Detection	Delivery Delay	Delivery Ratio	Message Sent
Interest	[32]	similarity among node Interests	infocom 06	SWIM	not addressed	✓		✓
	[36]	similarity among node Interests	PMTR	HCMM	Louvain algorithm	✓	✓	✓
	[11]	interest-based		CMM	Girvan-Newman			✓
	[34]	interest-based		CMM, RWP	not addressed		✓	✓
Flooding	[46]	flooding-based		ad hoc simulation	not addressed	✓		✓
	[24]	location-based		ad hoc simulation	not addressed			✓
	[1]	flooding-based		ad hoc simulation	not addressed	✓		✓
Social	[14]	general-purpose framework	infocom 06, MIT Reality, UCSD	not addressed	not addressed	✓	✓	✓
	[50]	publish/subscribe within communities	MIT Reality, UCDS, CAM, WirelessRope	not addressed	K-Clique, SIMPLE	✓	✓	
	[19]	centrality degree of nodes	infocom 06, MIT Reality, Cambridge, Hong-Kong	not addressed	K-Clique, Newman WNA		✓	✓
	[27]	measure of the duration of the enconters among nodes	NUS, infocom 06 MIT Reality	no	yes, assume the existence			

TABLE 1
Comparative tables of discovery strategies.

detection algorithm, the kinds of mobility traces used for the experimentation, and the metrics investigated.

3 REFERENCE SCENARIO AND PROBLEM DEFINITION

We consider an application scenario typical of opportunistic networking, where communications among people happens unpredictably, if and when they meet. As in Delay Tolerant Networks (DTN [15]) a person that has information to transmit carries it stored in the device cache until a communication opportunities arise, i.e., a wireless link with the device of another person can be established. At that point, the information is transferred (store-carry-and-forward). Because of the unpredictable nature of opportunistic encounters, these communications are frequently subject to high delays. Furthermore, differently from more general ad hoc networking [3], links are often asymmetric.

People move according to objectives and activities arising from their social relationships [7, 31]. In particular, human mobility is characterized by three key-aspects: *(i)* The mobility of a user is determined by her/his the social relationships [39]. For example, people with acquaintances located in a large urban area tend to move more frequently than more solitary individuals.

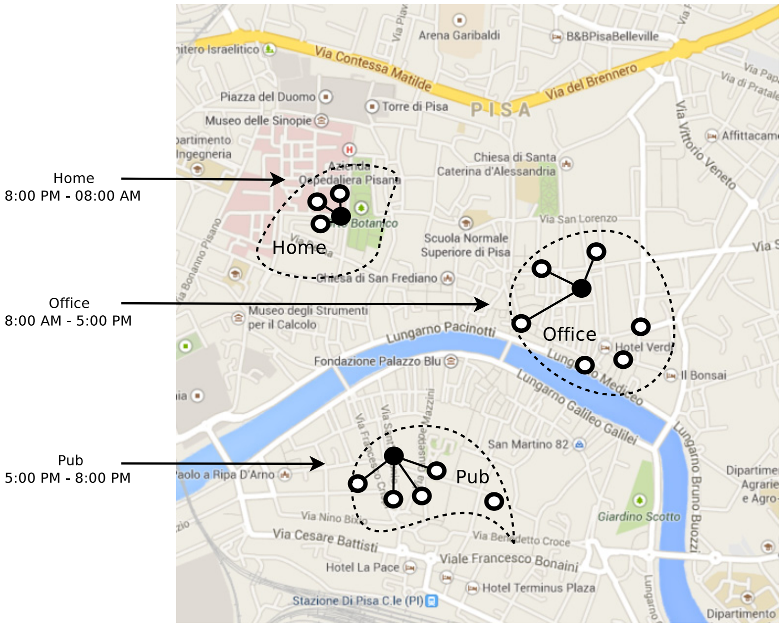


FIGURE 1
Example of the structure of the communities during a day.

(ii) The number of visited location is fairly limited (e.g., home, the office, a park or the supermarket) [22], and *(iii)* people tends to travel short paths instead of long routes [38]. Furthermore, people engaging in common activities tend to meet (i.e., tend to be in physical proximity) for longer periods of time: These people form a *community*. Examples of communities include colleagues who meet at the work place during regular working hours, or family and/or friends who meet after hours, at a local bar, pub, or at home.

Figure 1 shows three communities to which the person Alice (depicted as a full black circle) belongs. Throughout the day, Alice joins three different communities: Office (from about 8AM to about 5PM), pub (after hours) and home (night hours). Within each community, Alice meets colleagues and friends (indicated by hollow circles). At some times she is able to communicate with some of them (wireless links indicated by undirected lines), and not able to exchange information with some others (people outside of Alice's device transmission range).

People in the same community share similar *interests*. For example, the office community shares interests in working activities; members of the pub community are interested in sport, news and drinks; people at home share interests for recreational events, TV programs, etc.

Reminiscent of the *smart environments* of Weiserian memory [16, 47] a MSN enabling members of a community to communicate, also provides a wide range of *services* that people might want to access. Services are provided by the very people carrying mobile devices, such as smart phones and tablets. Examples include tethering for Internet access, sharing of photos and videos, of news, alert dissemination and weather forecast. A service is classified according to the features it provides.* As an example, services providing news, forecasts or stock market performance can be classified as *informative* services. Services providing music and video streaming contents are classified as *entertainment* services. The classification of a service is used by members of a community for discovering and querying for services matching with their interests.

4 PRELIMINARIES AND DEFINITIONS

In this section we provide notations and definitions used in the description of SIDEMAN.

4.1 Mobile social networks and community detection

A MSN is seen as a set of mobile nodes, each representing an individual moving within a bounded region. Nodes move driven by one or more objectives, e.g., traveling to the office, going back home or meet friends out. Occasionally, a nodes establishes a contact with another node by using a wireless communication interface (e.g., Bluetooth or WiFi). We model the connections among the nodes at time t as a directed graph $G_t = (V, E_t)$, where $V = \{n_1, n_2, \dots, n_v\}$ is the set of the v nodes of the MSN and $E_t = \{e_{ij} = (n_i, n_j) : n_i, n_j \in V\}$ is the set of links among the nodes at time t. Links in E_t can be directed, indicating one-way communications, typical of opportunistic networks. The neighborhood N_t^i of node n_i at time t is the set of nodes $n_j \in V$ such that $e_{ji} \in E_t$. In other words, the neighborhood of node n_i is made up of all nodes that can communicate with n_i at time t.

As a node n_i moves and comes into contact with other nodes, it keeps a *contacts history* for every node it connects with. The contact history stores some temporal properties regarding the node encountered, for example the average time between two consecutive encounters, the last time a node has been encountered or the average duration of an encounter. The contact history of node n_i is represented by the table T^i, used to detect communities.

* There are well-known methods for service classifications, relying on syntactical or ontology-based techniques [35]. This kind of classifications is beyond the scope of this work. For the purpose of demonstrating SIDEMAN, we use a simple syntactic association between services and interests, and interests and users.

In order to determine the community C where node n_i resides, at time t node n_i executes a community detection algorithm A using its contact history T_t^i and its neighborhood N_t^i at time t as input. The community C is a subset of node n_i current neighborhood N_t^i. The selection of which neighbors are part of the community of node n_i depends on algorithm A. SIDEMAN is independent of the specific algorithm used to detect communities, and can use any of the detection algorithms proposed for MSN [8, 20].

Node n_i keeps track of the communities it has been a part of in a table CT^i. Such table contains all the communities visited in time. Every time a community C is detected (by running algorithm A) it is inserted in CT^i only if there is no other community C' in CT^i that is *similar* to C. To determine similarity among communities, we use the Jaccard index [40] J, as commonly done by community detection algorithms [8]. Observe that $J \in [0, 1]$; if $J = 1$ then the intersection among community members coincides with the union of C and C', and hence C and C' are identical. If $J \geq \tau$ then C and C' are similar in the sense that they share a certain number of nodes. In both cases ($J \geq \tau or J = 1$) node n_i already visited C and it does not need to store C in CT^i again. Differently, if $J < \tau$ then community C is new and n_i has to store C in CT^i .

4.2 Service discovery

Nodes in a MSN provide services or collect information that they can share as services. In time, they need to announce these services in order to allow other nodes to discover and invoke them. We denote with $\mathcal{S} = \{s_1, \ldots, s_m\}$ the set of all services provided by the nodes in the MSN, $|S| = m$. Each service $s_j \in S$ has a set of service features describing some functional or non-functional properties of s_j. For instance, a feature may describe the interests associated to a service (e.g., sport, entertainment or news), another concerns Quality of Service (QoS) parameters such as latency or average response time, and another provides information regarding how to invoke the service provider. We assume that every service comprises at least the following:

A *service identifier* ID_j, which is the unique identifier of s_j.
Service interests SI_j, which is the set of interests associated to service s_j. The interests are labels tagging a service in terms of a common classification. For example the interests associated to a service for sharing media contents might be: *Media*, *entertainment*, *social*, etc. We assume that the set of all interests in the MSN is denoted with the set $\mathcal{I} = \{t_0 \cdots t_k\}$ where $|\mathcal{I}| = k$. The interests of service s_j are such that $SI_j \subseteq \mathcal{I}$.

Node n_i stores the services received by other nodes in its service cache $\mathcal{A}_i$. When searching for services matching some of its interests, node n_i checks its service cache. If the cache does not contain any service matching its interests,

n_i forwards a service query to the members of its community. A service query q contains a set of interests tagging the service $s_j \in \mathcal{S}$ needed by node n_i. Queries from node n_i that are still unanswered are kept in a pending query set PQ_i. Finally, we stipulate that every node n_i in the MSN is assigned some interests. The interests of node n_i are denoted by the set $I_i \subseteq \mathcal{I}$.

5 THE SIDEMAN ALGORITHM

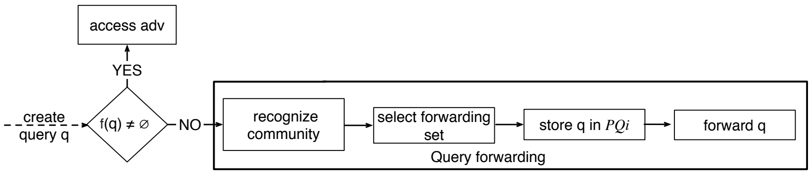
5.1 Overview of the algorithm

SIDEMAN is an algorithm used by a node n_i for discovering services available in a MSN. To this purpose, SIDEMAN enables nodes (i) to reactively disseminate queries for services they do not currently have, (ii) to proactively exchange services they might be interested in, so that a node has that service when it needs it, and (iii) to manage the reception of queries and services. The design of SIDEMAN exploits a typical feature of MSNs: People tend to form communities made of similar individuals (Section 3). In particular, SIDEMAN relies on the detection of communities for optimizing operations (i) and (ii) as follows:

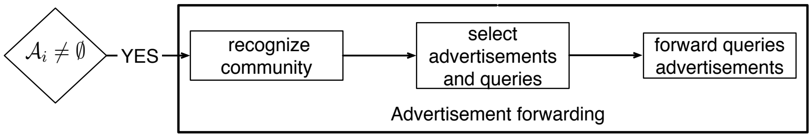
- The dissemination of queries is realized only among members of the same community, thus avoiding their indiscriminate flooding through the whole neighborhood.
- The exchange of services is performed only inside a community. The rationale of this choice is that since a community is often formed by similar individuals, it is more likely to find services of interest inside the community.

To the best of our knowledge, these are innovative aspects informing the design of SIDEMAN that are not considered in previous works.

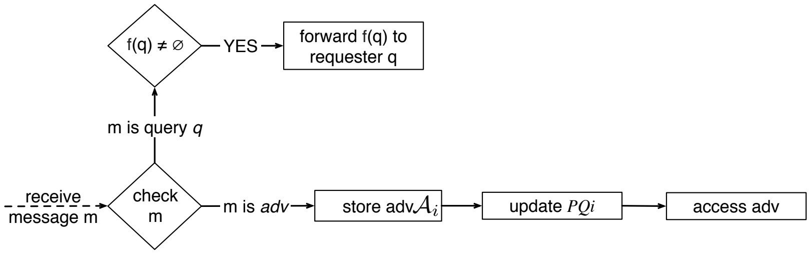
Whenever a node needs a service (reactive phase of SIDEMAN), it crafts the query q and it checks if its cache stores a services matching with q (Figure 2(a)). If the node finds a service matching q then it accesses the service provider and it invokes the service (the service access phase is out of the scope of this work, since it can be implemented with specific protocols for service access). To this purpose, we define the function $f(q)$ that given a query q it returns all services matching q that are stored in the service cache of the node running f. Otherwise, n_i runs an algorithm for detecting the community to which it belongs. Once a community is detected, n_i checks for a *similar* community in its community table. If such a community exists, i.e., if n_i has already visited that community in the past, n_i *recognizes* it. Otherwise, n_i stores the new community in its community table.



(a) Reactive phase


$\mathcal{A}_i \neq \emptyset$
YES
recognize community
select advertisements and queries
forward queries advertisements
Advertisement forwarding

(b) Proactive phase



(c) Message reception phase

FIGURE 2
Overview of SIDEMAN.

After the community detection phase, n_i builds a forwarding set from members of its current community whose interests match those in q. In other words, the query q is forwarded only to the members of the community of n_i sharing at least one interest with q. At this point, node n_i stores the query q in the set of its pending query, such set represents the queries that are still waiting for an answer.

At regular intervals, node n_i executes the proactive component of SIDEMAN, for exchanging services with members of its community (Figure 2(b)). If n_i carries a service whose interests match those of at least one member of its community, then n_i exchanges such service. During this phase, node n_i also forwards the queries left in PQ_i to the member of the community previously detected.

Figure 2(c) depicts the situation in which node n_i receive a message (either query or service) from another node. If the message received is a query q, node n_i checks if it can provide an answer to q. If this is the case, node n_i replies to the requester node with the set of services matching q. Otherwise, if the message received is the service s_j, then n_i stores s_j in $\mathcal{A}_i$ and removes all the pending queries answered by the service s_j.

5.2 SIDEMAN

Before describing the reactive and proactive phases of SIDEMAN (Figure 2(a) and 2(b)), the following algorithm describes how n_i recognizes the community C.

A community C is determined by running any community detection algorithm A [20] (line 1). A community C is *recognized* if there is a community C' in CT^i (the community table of node n_i) that is *similar* to C according to a given similarity index. As mentioned, we used the Jaccard index with threshold τ [40]. If a community C' is found that is Jaccard-similar to C (i.e., $J(C, C') \geq \tau, where J is the Jaccard index$) then the set of interests $\mathcal{I}_C$ of community C are those of community C' (line 3). If no such a community exists, the interests of the members of the community must be collected by asking to every community member its interests. This set of communication is performed through executing the function $getInterests(\cdot)$ (line 5). In particular the $getInterests(\cdot)$ function iterates over the members of C and asks to every member the list of its interests. The recognized community C is finally returned (line 7).

We stipulate that Algorithm 1 returns the pair $< C, \mathcal{I}_C >$, where C is the community (the set members of the community) and $\mathcal{I}_C$ is the list of interests shared by the members of the community. We recall that the community table CT^i is used for storing the communities visited by n_i and the interests of the community members. Through this table we can drastically reduce the number of times the function $getInterests(\cdot)$ is invoked by n_i. Our experiments show that communities are recognized always over 50% of the times, and that, at steady state, the function $getInterests(\cdot)$ is called in about half of the cases.

The reactive phase of SIDEMAN is implemented by Algorithm 2.

Node n_i starts by checking whether it has a service matching the query q (line 1). If $f(q) \neq \emptyset$ (where f is the function for checking the existence of

Algorithm 1 Community detection

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

Algorithm 2 Reactive phase

```
1: if $f(q) \neq \emptyset$ then
2:     $s_j = f(q)$
3:     access $s_j$
4: else
5:     $< C, \mathcal{I}_C >=$ RecognizeCommunity($T^i, N^i, CT^i, \tau$)
6:     $V_q = \{n_j \in C \mid \exists\, t_k \in \mathcal{I}_q$ s.t. $t_k \in I_j\}$
7:     Forward $q$ to $V_q$
```

services matching with q), then n_i accesses the service matching with q. Otherwise, n_i recognizes the local community to which it belongs (Algorithm 1). Node n_i then computes the set $V_q \subseteq C$ of nodes that can potentially answer query q (line 6). (Set V_q is composed by nodes in C that share at least one interest with q.) Finally, the query is forwarded to nodes in V_q (line 7). We observe that a naïve implementation of the creation of the set V_q (based on checking that the interests in q also belong to the interests $\mathcal{I}_j$ of node n_j) may require an order of complexity $O(|C| \cdot |\mathcal{I}_j| \cdot |\mathcal{I}_q|)$, due to the need for finding a match, for each community member, between its interests associated to the query. We significantly reduce this complexity by using Bloom Filters to implement the set $\mathcal{I}_j$ [42]. A Bloom Filter is a data structure along with two core operation for checking whether or not an element belongs to the data structure (membership), and for adding an element to the data structure (addition). Both operations are performed in constant time thanks to use of h distinct hash functions [42]. In our case, we used Bloom Filters to check if t_k is in $\mathcal{I}_q$ and to check if t_k is in $\mathcal{I}_j$ (line 6 of Algorithm 2). The complexity of creating V_q can be reduced to $O(|C|)$.

The proactive component of SIDEMAN that takes care of service exchange and forward of pending queries is performed at node n_i by executing the following Algorithm 3.

Node n_i starts with recognizing its community. To this purpose it uses Algorithm 1 described above that returns C as well as $\mathcal{I}_C$.

For every interest $t \in \mathcal{I}_C$ node n_i performs the following actions.

1. It computes the set $V_t \subseteq C$ of nodes in C also interested in t and the set $S_t \subseteq \mathcal{A}_i$ of services matching t (lines 3 and 4). If V_t is empty, n_i removes t from $\mathcal{I}_C$ (line 6). In this way, if the same community is recognized at a later time, a node can avoid considering interests that are no longer shared by the members of C.
2. It forwards the set S_t to nodes in its community sharing the same interests (line 7).

Algorithm 3 Proactive phase

```
1: $< C, \mathcal{I}_C >=$ RecognizeCommunity($T^i$, $N^i$, $CT^i$, $\tau$)
2: for all $t \in \mathcal{I}_C$ do
3:     $V_t = \{n_j \in C \mid t \in \mathcal{I}_j\}$
4:     $S_t = \{s_j \in \mathcal{A}_i \mid t \in SI_j\}$
5:     if $V_t = \emptyset$ then
6:         $\mathcal{I}_C = \mathcal{I}_C \setminus \{t\}$
7:     Forward $S_t$ to $V_t$
8:     for all $q \in PQ_i$ do
9:         if $t \in q$ then
10:            Forward $q$ to $V_t$
```

3. It selects those pending queries from PQ_i that concern interest t, and forwards them to members of its community sharing that interest (line 10).

As for Algorithm 2, the constructions of sets V_t and S_t in Algorithm 3 are implemented through Bloom Filters. In this way, the complexity of computing V_t and S_t is $O(|C|)$ and $O(|\mathcal{A}_i|)$, respectively.

Whether reactively sending out service queries (Algorithm 2) or proactively exchanging services and disseminating pending queries (Algorithm 3), some of the node in the neighborhood of n_i might transmit responses. The following Algorithm 4 describes node n_i reaction to receiving response m.

Upon receiving response m, node n_i checks whether it is a query or a service. If m is a query then the node determines all its services that answer that query, if any. These services are then sent directly (i.e., through a unicast transmission) to the node n_r requesting the query m. If m is instead a service, node n_i updates its service cache $\mathcal{A}_i$. Specifically, the function Update($\mathcal{A}_i$, m) checks whether m is already in $\mathcal{A}_i$. If this is the case, the corresponding entry is updated. Otherwise, m is added to the cache.

Algorithm 4 OnMessageReception(m)

```
1: if $m$ is a query then
2:     $n_r$ is the requester of $q$
3:     $D = f(m)$
4:     Forward $D$ to $r$
5: else
6:     Update($\mathcal{A}_i$, $m$)
```

6 PERFORMANCE EVALUATION

We evaluate the performance of SIDEMAN through Java-based simulations that take into account characteristics of MSN. We implemented SIDEMAN as well as two algorithms for service discovery in MSN, namely, the "social" version of algorithms for flooding [26] and gossiping [25]. In this section we describe in detail the mobility model and mobility traces we used for node mobility, the simulator and its parameters, the metrics of interest chosen for comparing the performance of the three algorithms, and the simulation results.

6.1 Simulation scenarios

We consider two MSN scenarios where $v = 78$ nodes move in a square area of given side. The scenarios differ depending on the mobility of the nodes.

In the first scenario nodes move according to the Home-cell Community-based Mobility Model (HCMM) [7], defined to mimic human mobility. We consider three different deployment areas, small, medium and large, with side of 800m, 1400m and 2000m, respectively. Each area is configured as a grid made up of 5 cells [7]. Every node is associated to a home cell. Nodes in the same home cell share social ties. Some nodes also have social ties with other nodes from different cells. These nodes are called traveler nodes. The strength of the social ties of the travelers is determined by the so-called rewiring probability. The rewiring probability models the relationship between nodes of different cells and drives the mobility of nodes. The mobility speed is typical of pedestrian walking, i.e., from 1 to 1.86m/s [7]. We call this scenario the HCMM scenario.

The second scenario we consider is based on real traces gathered from participants to the IEEE Infocom conference from April 24 to April 27 2006 [19]. In that scenario 78 conference attendants were given Intel iMote devices equipped with a Bluetooth transceiver with a transmission range of about 30m. Traces were collected each day from 7.00AM to 9.00PM. Each participant filled a survey about her/his interests, nationality, language spoken, etc. (We use this survey to assign interests to the nodes). This second scenario is referred to as the Infocom 06 scenario.

6.2 Simulator and parameters

Our Java-based simulator implements three important components of service discovery in MSN: (i) Contact history maintenance, (ii) community detection, and (iii) the service discovery algorithms. (i) Contact history maintenance concerns the contact history at each node, storing parameters such as those described in Section 4. (ii) Community detection is implemented by a well known solution, called AD-SIMPLE [8]. AD-SIMPLE is an enhanced version of SIMPLE, an algorithm for community detection that has been shown to outperform previous solutions such as k-CLIQUE and MODULARITY [20]. The reasons for this choice are multi fold: First of all, AD-SIMPLE is a distributed algorithm, thus being suitable to run in mobile networks of small, resource-constrained devices such as MSNs. Secondly, despite being distributed, the communities it detects are very similar to those detected by SIMPLE [8]. In fact, the similarity index between the communities detected by AD-SIMPLE and SIMPLE is far higher than that of the communities detected by k-CLIQUE and MODULARITY and their distributed counterparts. Finally, AD-SIMPLE implements an effective mechanism for removing nodes from a community that they have not joined for a while. In our simulation we stipulate that two communities are recognized as the same community is their similarity index τ is ≥ 0.8 (Section 4). Finally, (iii) service discovery is implemented by SIDEMAN (Section 5). This last point is very significant since SIDEMAN, s-Flooding and s-Gossip all are community-based.

Node behavior is determined by considering query generation rate, service generation rate and the distribution of interests to nodes. The query generation rate determines how many queries are generated by the nodes. We modeled this rate as a Poisson process of intensity λ queries per second. In our simulations we set $\lambda = 3$. In particular, every second a number q of query is generated and assigned to q nodes selected randomly and uniformly among all nodes. The service generation rate concerns the services the nodes store in their service cache without using the service discovery algorithm. Initially, a node cache is empty. In time, services are assigned to the nodes. The service generation rate is also a Poisson process of intensity μ services per second. The value of μ is set to 3. Every second a number r of services is generated and assigned to r nodes selected randomly and uniformly among all nodes. Each time the r services are drawn randomly and uniformly from a set $\mathcal{S}$ of m services. The value of m is set to 10000.

Interests distribution depends on the scenario. In the HCMM scenario interests are given to nodes as follows. Every cell x is assigned k_x interests according to a Zipf's distribution (with parameter skew $= 1$). The parameter k_x is the ratio between the total number n of interests in the simulation (set to 35 in our simulations) and the number of cells in the HCMM grid. In our experiments $k_x = \frac{35}{5} = 7$. The interests assigned to cell x are denoted with the set $\mathcal{I}_x$. Node n_i is assigned $|\mathcal{I}_i|$ interests from $\mathcal{I}_x$ proportionally to the time it spends in cell x. In this way, nodes visiting the same cell for a long time share similar interests. The values of λ and μ, as well as that of m, n and k_x have been selected consistently with those of similar scenarios in previous works [5, 6, 28].

	Infocom 06	**HCMM**
Simulation area side	conference room	800m, 1400m, 2000m
Transmission range	30m	30m
Node speed	n.a.	from 1 to 1.86m/s
Simulated time	$t = 201600$s	$t = 300000$s
Number of nodes	$v = 78$	
Service interests	$n = 35$	
Services	$m = 10000$	
Query rate	$\lambda = 3$	
Service rate	$\mu = 3$	
Community similarity	$\tau = 0.8$	
Community recognition	AD-SIMPLE	
Frequency of proactive phase	600 s	

TABLE 2
Simulator parameters.

In the Infocom 06 scenario we use the association between interests and nodes that comes with the traces.

The number of messages exchanged by a node upon encountering another node, and the corresponding energy consumption, is computed according to the data sheet of the WiFi/ Bluetooth chip Broadcom®BCM4330 of the Samsung Galaxy S III smart phone.

Our simulation results for the first scenario are obtained by averaging the outcomes of 1000 runs, each running for 300000s (around three days and a half), each time on a different HCMM trace. This number of experiments achieves a statistical confidence of 95% within a 5% precision. Since we are interested in steady state performance, all metrics have been collected after 10000s from simulation start, when we observed that nodes have made a consistent number of contacts and the node cache size stabilizes. Table 2 summarizes the simulation parameters.

6.3 Benchmark algorithms

To demonstrate its effectiveness in discovering and advertising services, we compared the performance of SIDEMAN against that of two algorithms for service discovery in wireless networks, which we modified to work in MSNs. The two algorithms, termed s-Flooding and s-Gossip, are the ”social” version of traditional flooding [26] and gossiping [25] algorithms. By using traditional flooding a node would spread queries and services among all its neighbors. Through gossiping a node would instead send those queries and services only to a random subset of its neighbors. The social component is introduced in s-Flooding and s-Gossip by having nodes exchanging services and queries within communities, rather than within their neighbors.

	Select forwarding set	**Select services and queries**
SIDEMAN	Members of the current community with interests matching the query.	Services and queries whose interests match those of the community members.
s-Flooding	All members of its current community.	All services and queries to all community members.
s-Gossip	Random number of members of its current community.	Services and queries to a random number of members of the current community.

TABLE 3
Benchmark algorithms.

The behavior of s-Flooding and s-Gossip is similar to the one given for SIDEMAN (see Figure 2), in particular the two algorithms are composed by a reactive phase or discovering the service matching with a given query, a proactive phase whose is disseminating queries ad services stored at a node, and a message management phase for managing the reception of queries and services. The main differences of s-Flooding and s-Gossip with respect to SIDEMAN are the implementations of the *select forwarding set* of the Reactive phase (Figure 2(a)) and *select services and queries phase* of the Proactive phase (Figure 2(b)). Table 3 shows the main differences among the three algorithms.

We choose s-Flooding and s-Gossip for three reasons: (i) They represent useful performance benchmark for service dissemination in MSNs, (ii) they implement a simple but effective strategy for the propagation of information, being both designed to maximize the number of queries and services exchanged among nodes, and (iii), similarly to SIDEMAN, they are implemented for propagating messages (either queries and services) only inside a community, without flooding the whole network.

6.4 Evaluation metrics

We compare the performance of SIDEMAN, s-Flooding and s-Gossip with respect to the following metrics.

- *Recall*, defined as the ratio between the number of services stored in the cache of node n_i that are of interest to n_i and the total number of services stored in its cache. This metric measures the effectiveness of a service discovery algorithm in propagating only those services that are of interest to nodes and in avoiding to disseminate services that a node would not use. In this way, traffic overhead as well as energy consumption are minimized. Clearly, the value of R is in [0, 1]. A value of 0 means that none of the services in which node n_i is interested are in its service cache (worst case). A value of 1 indicates the best case: Node n_i stores only services in which it is interested.

- *Gain*, defined as the ratio between the number of times a node finds service s already in its service cache and the number of times a node has to query for it. This metric indicates the effectiveness of a service discovery algorithm in reducing the number of queries to obtain a service. Clearly, the value of G is in [0, 1]. A value of 0 means that every time a node needs a service it has to query for it (worst case). A value of 1 indicates the best case: Every time a node needs a service, that service is in its cache.
- *Energy Cost (EC) per node*, defined as the average energy consumption incurred by each node during network operations. This metric provides an indication of the impact of running a service discovery algorithm on a device batteries. The lower the EC, the better the energy savings and the performance of the algorithm. EC is expressed in Joules. Its computation does not consider the community detection algorithm, thus indicating only the energy consumption imposed by running a service discovery algorithm.
- *Query Response Time* (in seconds), defined as the average time elapsed between when a query is sent and the reception of the first service matching that query. This metric informs us about how effective a discovery algorithm is letting a node receive the service it needs swiftly.
- *Service Cache (SC)*, defined as the average number of services that a node stores in its service cache in time. This metric informs us about the memory space needed by a node for running the discovery algorithm.
- *Services Exchanged (SE)*, defined as the average number of services that are exchanged by a node with other nodes. This metric indicates the number of interactions that every node performs in time.
- *Network Overhead*, defined as the average number of packets that a node exchanges by running each of the discovery algorithms. This metrics provides an indication of the overhead introduced by each of the discovery algorithms in terms of network operations.

6.5 Simulation results

This section shows the performance of the service discovery algorithms that we compare: SIDEMAN, s-Flooding and s-Gossip. We depict the results for each algorithm as time progresses, to provide an indication of their effectiveness in letting nodes converge to a state in which most of the services

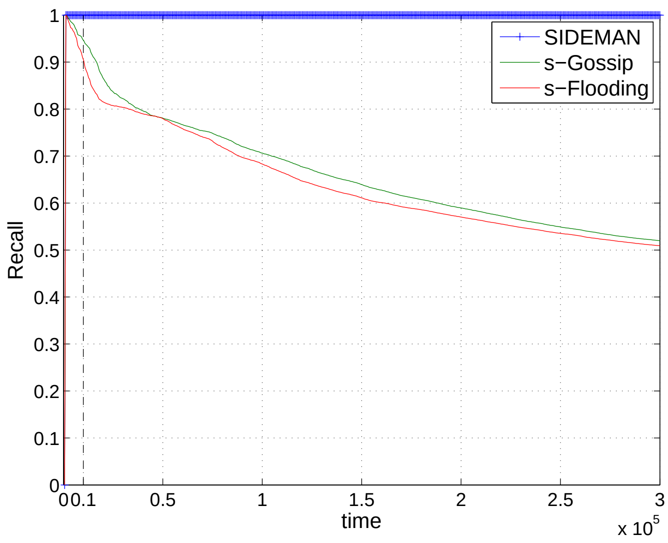


FIGURE 3
Recall in the HCMM scenario.

they need are in their cache. We start with results for the HCMM scenario, followed by those for Infocom 06.

HCMM scenario

Results concerning the investigated metrics are shown in figures 3 to 6. The figures refer to 78 nodes roaming in a $800 \times 800\text{m}^2$ area. (Results for nodes traveling in bigger areas show similar trends. Their values are commented in details at the end of the section.)

Recall. Results concerning Recall are shown in Figure 3. SIDEMAN always obtains a Recall equal to 1, meaning that throughout network operations nodes store in their caches all and only services that they are interested in. This is a consequence of the very nature of SIDEMAN service exchange according to which a service is sent to a node only if that node is interested in it.

The recall of s-Flooding and s-Gossip instead decreases in time, reaching a value as low as 0.5 by the end of the observation period. In other words, half of the services stored in the service cache of nodes running s-Flooding and s-Gossip will never be used by those nodes. This is because s-Flooding and s-Gossip service exchange is not interest-based. We notice that s-Gossip slightly outperforms s-Flooding. This is because a node running s-Gossip sends services to a number of nodes that is lower than the number of nodes involved in service exchange in s-Flooding.

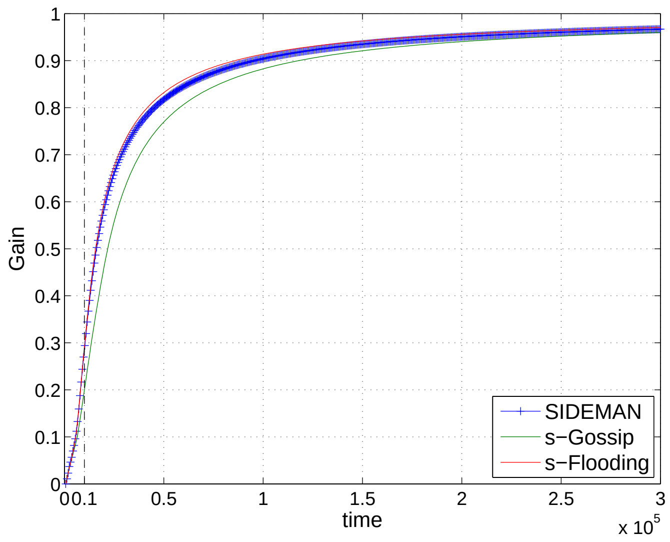


FIGURE 4
Gain in the HCMM scenario.

Gain. Results for the Gain achieved by the three algorithms are shown in Figure 4. The trend is similar for SIDEMAN, s-Flooding and s-Gossip. In particular, with passing time the Gain increases and tends to its maximum 1. (By the end of the observation period it is 0.96 for each of the three algorithms.) As expected, the Gain increases rapidly during the initial time interval $[0, 0.5 \cdot 10^5]$s because nodes start with no services in their cache and then they begin moving and exchanging services as they meet and form communities.

By the end of the observation time they have exchanged enough services to find the service they need in their cache. Results about this metric show clearly how effective the strategy followed by SIDEMAN is for service distribution. The Gain of SIDEMAN is never noticeably lower than that of s-Flooding and s-Gossip, two algorithms that are designed to maximize the diffusion of information (i.e., services, in our case). In fact, we notice that the Gain of SIDEMAN is always (slightly) better than that of s-Gossip throughout the network operation time. This is because, despite the number of services distributed by s-Gossip is far higher that that of those distributed by SIDEMAN, most of these services are not of interest to the querying node.
Energy cost. In terms of energy cost SIDEMAN remarkably outperforms s-Flooding and s-Gossip (Figure 5). The energy consumption of SIDEMAN is at least 7 times lower than that of s-Flooding, and 6 times lower than that of s-Gossip. (The energy cost of s-Gossip is slightly lower than that of s-Flooding

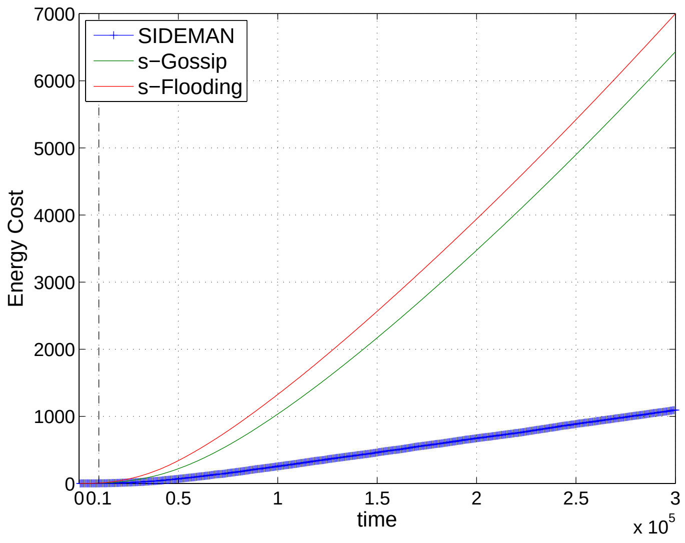


FIGURE 5
Energy cost in the HCMM scenario.

because, as mentioned already, for its very nature s-Gossip exchanges less services that s-Flooding.) To show the impact on device lifetime of running the three algorithms, we considered nodes powered by a standard battery pack with an average consumption of 7.9Watt/hour (i.e., a battery pack with a capacity of 2100mAh and voltage of 3.8V). By the end of the observation time devices running s-Flooding have consumed an average of 87.7% of their initial battery charge, devices running s-Gossip have consumed 80.2% of it, while devices running SIDEMAN have consumed only 12.5% of the initial battery charge.

Query response time. Results for the query response time of the three algorithms are shown in Figure 6. We notice that the time needed to respond to a query grows in time. This is because as time progresses, nodes tend to visit roughly the same communities all over again, and if a node does not find a service in its cache right away, and has to query for it, it is unlikely that it will receive this service from the nodes that it has already met and keeps meeting. In time, it might enter a new community whose member might have the required service, but that happen with low probability. (These probabilities are confirmed by an in-depth study of community properties, not shown here.) The performance of SIDEMAN is sandwiched between that of s-Flooding and s-Gossip. Since s-Flooding distributes the largest number of services inside a community, nodes carry more services around, increasing the probability of being able to respond to a query. That is why its query

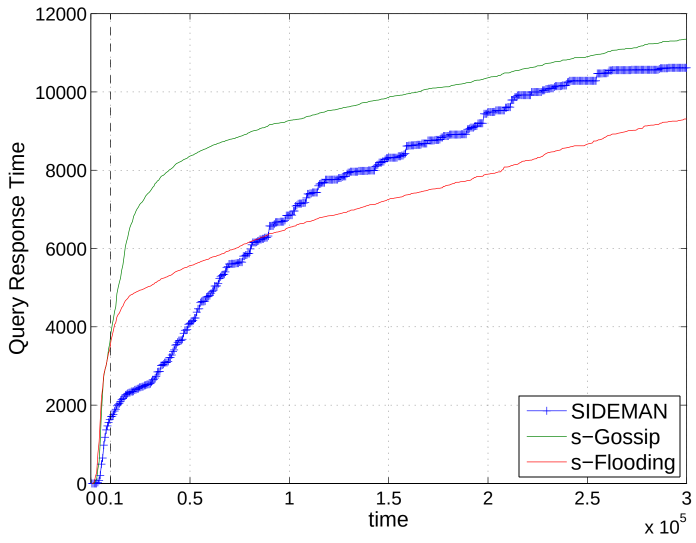


FIGURE 6
Query response time in the HCMM scenario.

response time is the lowest. The nature of s-Gossip is to distribute services to a subset of the nodes in a community selected randomly. The effect is that nodes exchange a lot of services in which they are not interested, and therefore the probability of a node to respond to a query decreases, increasing the response time. We observed that by the end of network operation, SIDEMAN has exchanged 84.32% (82.84%) less services than s-Flooding (s-Gossip). However, since nodes running SIDEMAN exchange only those services in which they are interested, the probability of a node to find a service in its cache is fairly high, as confirmed by checking the number of queries sent by nodes throughout the observation time: The average number of queries sent by a node running SIDEMAN is 9, while it is 54 and 63 for nodes running s-Flooding and s-Gossip, respectively. As a consequence the query response time for SIDEMAN soon becomes lower than that of s-Gossip.

Service Cache. Results on the average size of the service cache are shown in Figure 7. We observe that the dimension of the service cache for the three algorithms grows rapidly in the time interval $[0, 0.5 \cdot 10^5]$s because nodes start with empty caches and then they begin moving and exchanging services as they meet and form communities. After the initial phase the behavior of the algorithms are quite different. In particular, nodes running s-Flooding and s-Gossip keep increasing the dimension of the service cache as the simulation time passes. Nodes running SIDEMAN instead, show a better control of the size of the cache. The nature of s-Flooding and s-Gossip is to always

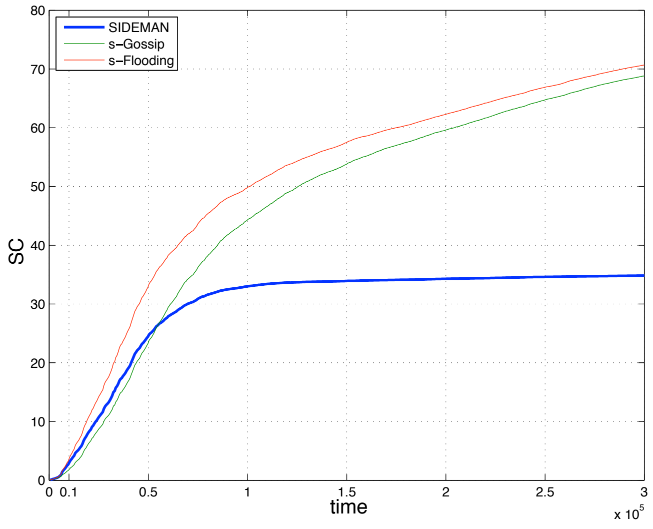


FIGURE 7
Service cache size in the HCMM scenario.

maximize the number of services exchanged, without considering if a service matches the interests of the node receiving it. By the end of network operation, nodes running s-Flooding and s-Gossip store respectively an average of 71 and 69 services, while nodes running SIDEMAN store an average of 35 services.

Services Exchanged. Results concerning the average number of services exchanged are shown in Figure 8. The number of exchanges grows in time, but the performance of the three algorithm is quite different. Nodes running s-Flooding and s-Gossip exchange a number of services that grows sub-linearly with respect to the observation time, while the exchange of services of nodes running SIDEMAN is always bound within a limited interval. The curves shown in Figure 8 are tightly coupled with those of the size of the service cache (Figure 7) and of the energy cost (Figure 5). In particular, lowering the number of exchanges also reduces the services stored in the cache and the energy cost due to exchange of services.

It is worth to notice that, by the end of the observation time, nodes running s-Flooding and s-Gossip exchange an average of 2.57×10^4 and 2.42×10^4 services. Considering that we considered scenarios with 10^4 services (Table 2), this shows that nodes running s-Flooding and s-Gossip exchange more than twice the number of services available. Conversely, nodes running SIDEMAN exchange an average of 430 services, which are those to which the nodes are really interested in.

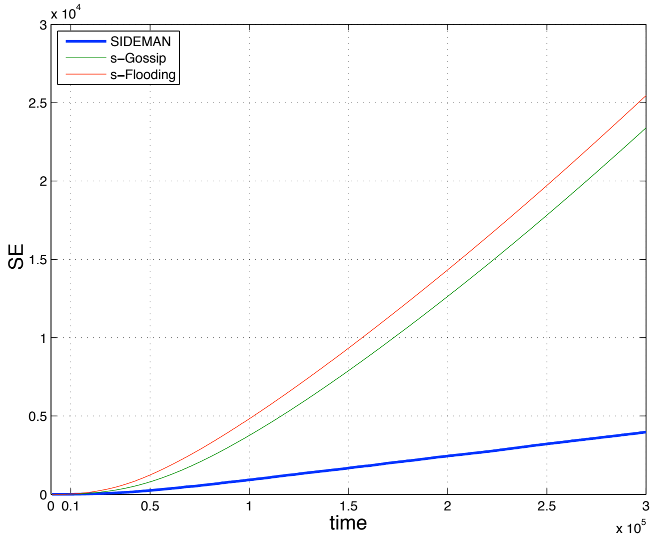


FIGURE 8
Number of services exchanged in the HCMM scenario.

Network Overhead. Results concerning the network overhead are shown in Figure 9. We observe that the average number of packets needed for a single run of SIDEMAN (comprising the reactive and proactive phases and the management of incoming messages) is lower than that of s-Flooding and s-Gossip. During the first part of the simulation $[0, 0.5 \cdot 10^5]$s the network overhead of the three algorithms is comparable, because nodes start with empty caches. As time progresses, nodes begin exchanging queries and services and the algorithm performance differ. In particular, we observe that s-Flooding is the algorithm with the highest network overhead because its flooding strategy (Table 3) requires to forward each query and each service to every member of the node community. Differently, s-Gossip incurs lower complexity than s-Flooding because queries and services are forwarded to a number of community members less than or equal to the size of the node community. SIDEMAN achieves the lowest network overhead along with perfect Recall and comparable Gain, which makes it suitable for application scenarios with nodes that are resource constrained have constrained.

Performance in larger deployment areas. We have also evaluated the performance of the three algorithms in larger deployment areas, namely, 1400m $\times$1400m and 2000m $\times$2000m. We observed that by increasing the dimension of the scenario without increasing the number of nodes they tend to travel shorter routes and restrict their mobility to few locations. Such a more limited mobility implies that a node enters in contact with a small sub-set

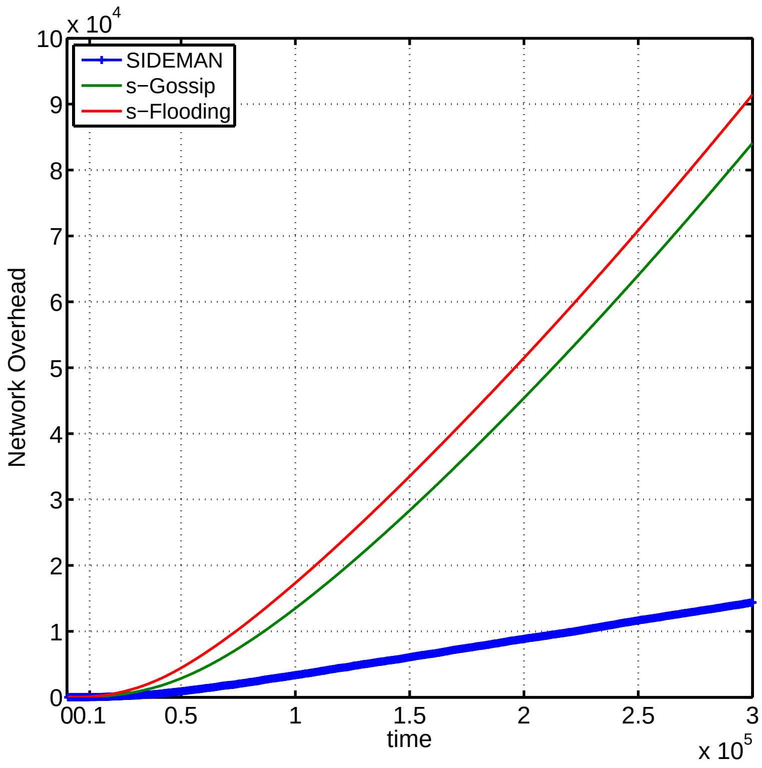


FIGURE 9
Network overhead in the HCMM scenario.

of the other nodes. In particular, we observe that nodes tend to form smaller communities with respect to scenarios in smaller areas Performance results are affected as follows: The Recall of SIDEMAN is equal to 1, irrespective of the size of the scenario. s-Flooding and s-Gossip show a Recall that decreases slower than that in the original scenario (Figure 3). This is because nodes encounter fewer nodes and therefore exchange fewer services. As a result the probability of storing services not of interests to the nodes decreases. The Gain of the three algorithms is slightly lower than that shown in Figure 4 (less than 2%). This is due, again, to the lower mobility of the nodes, which decreases the probability of exchanging services with other nodes, and therefore of the probability of a node of finding a service in its cache. As expected, the energy cost for larger scenario decreases, because of the lower number of services exchanged: The fewer the nodes exchange services and queries, the lower the energy consumption. The query response time increases with the size of the scenario, since decreased mobility leads to a slower propagation

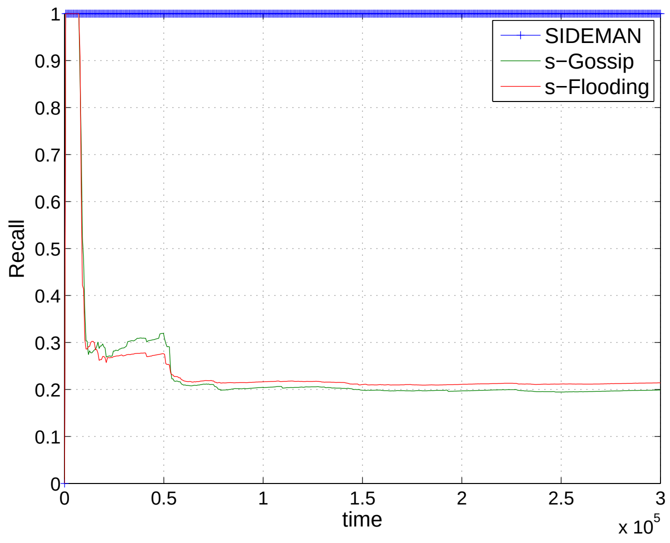


FIGURE 10
Recall in the Infocom 06 scenario.

of the services. Therefore, as soon as a node queries for a service it has to wait a longer time before receiving the matching service. Lastly, the average size of the service cache also decreases with increasing the size of the scenario, since low mobility reduces the probability of exchanging services and therefore also the probability of storing services in the cache.

Infocom 06 scenario

Figures 10 to 13 depict the results of SIDEMAN, s-Flooding and s-Gossip in the Infocom 06 scenario.

Recall. Results concerning Recall are shown in Figure 10. The Recall of SIDEMAN is 1, as expected. The Recall of s-Flooding and s-Gossip rapidly decreases to 0.2. This depends on the nature of the Infocom 06 traces, where users in the same community share a low percentage of interests (this percentage in time is as low as 0.8%). For instance, there are times when all participants are at a common event, such as a meal break, or a plenary session. In this case, the community is very large. The interests shared by the community members are instead very few. This is in contrast with the results from the HCMM scenario, where nodes in the same community would share instead up to 80% of their interests. This explains why, eventually, in the HCMM scenario s-Flooding and s-Gossip show value of Recall higher than those in the Infocom 06 scenario (i.e., around 0.5 vs. 0.2). In the Infocom 06 scenario, nodes running s-Flooding and s-Gossip are more likely to exchange

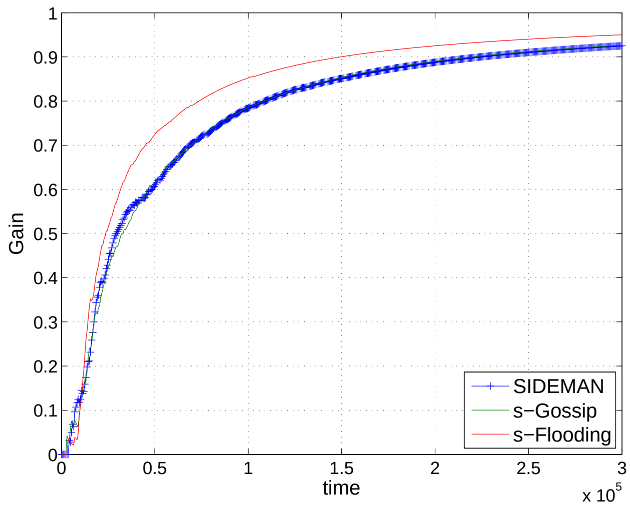


FIGURE 11
Gain in the Infocom 06 scenario.

services in which they are not interested. As a result, the Recall decreases more rapidly than that in the HCMM scenario.

Gain. Results for the Gain of the three algorithms are shown in Figure 11. The trend is the same for all algorithms and by the end of the observation time the Gain reaches the value 0.94. Similarly to Figure 4, the Gain increases rapidly in the interval $[0, 1 \cdot 10^5]$s because nodes start with no services in their cache and then they begin moving and exchanging services as they meet and form communities. By the end of the observation time, the nodes have exchanged enough services to find the service they need in their cache. Although showing the same trend observed for the HCMM scenario, the values of the Gain in the Infocom 06 scenario are lower than that for the HCMM one. This is because the communities at Infocom share fewer interests than those shared by the HCMM communities. As a consequence, when a node queries for a service, the probability of receiving a response to its query is low, and consequently the Gain grows more slowly.

Energy Cost. Figure 12 shows the energy cost incurred by running the three algorithms. As in the HCMM scenario, SIDEMAN outperforms s-Flooding and s-Gossip. In particular, the energy cost of SIDEMAN is far lower than that of the other two algorithms. If we consider a standard battery pack as described for the HCMM scenario, we observe that devices running s-Flooding and s-Gossip deplete their battery approximately after $1.5 \cdot 10^5$s (half of the observation time), while at the end of the observation time devices running SIDEMAN have consumed only 12% of their energy.

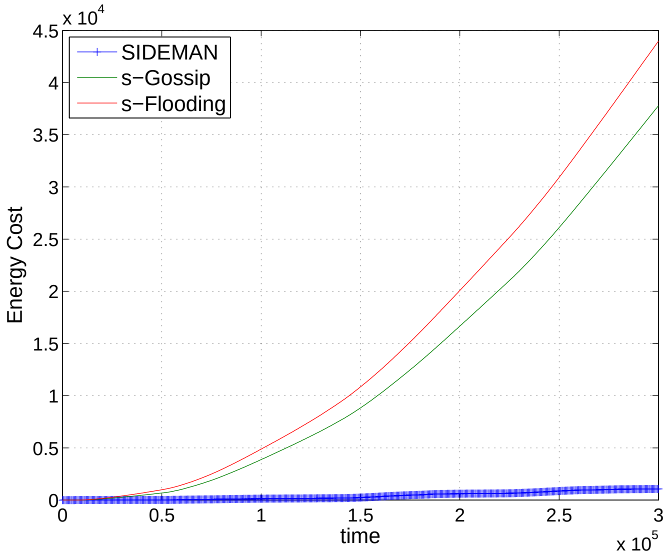


FIGURE 12
Energy cost in the Infocom 06 scenario.

Query Response Time. Results for the query response time of the three algorithms are shown in Figure 13. The time needed to respond to a query grows in time. As in the HCMM scenario, nodes tend to keep visiting the same communities. As such, if a node has to query for a service, it is unlikely that it will receive this service from the nodes that it has already met and keeps meeting. In time, a node might enter a new community whose members have the required service. As mentioned, this happen with low probability. The performance of SIDEMAN is similar to that observed in the HCMM scenario: The query response time is sandwiched between that of s-Flooding and s-Gossip. The reasons are the same explained in Section 6.5. However, the values of the query response time in the Infocom 06 scenario are far higher that those of the HCMM one: In fact, they double. As discussed for Recall (Figure 10) nodes in the same community share few services, and therefore if a node does not find a service in its service cache, it is unlikely that it will receive it from another node currently close.
Service Cache. Results for the average size of the service cache are shown in Figure 14. The service cache grows similarly to what observed in the HCMM scenario. In particular nodes running s-Flooding and s-Gossip have the highest dimension of the cache. In this simulation scenario, nodes running the three algorithms (s-Flooding, s-Gossip and the SIDEMAN) store a number of services higher than that in the HCMM scenario. For example, in the

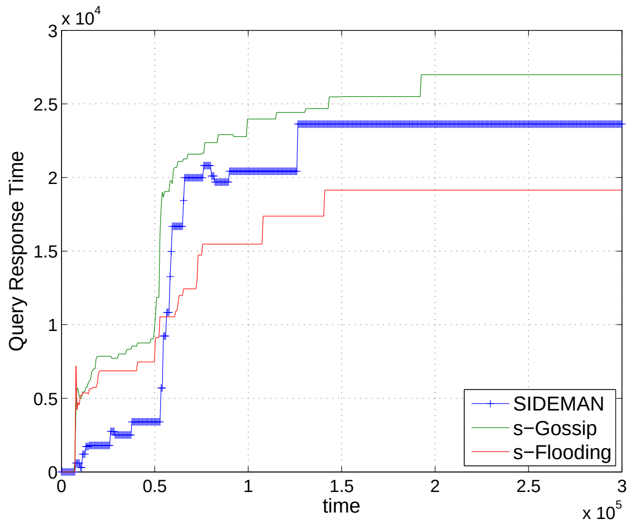


FIGURE 13
Query response time in the Infocom 06 scenario.

Infocom 06 scenario nodes running s-Flooding and s-Gossip store an average of 600 and 530 services, respectively, and an average of 71 and 69 services with the HCMM scenario. This behavior depends on nature of the mobility traces used in the Infocom 06 scenario, where nodes tend to encounter many

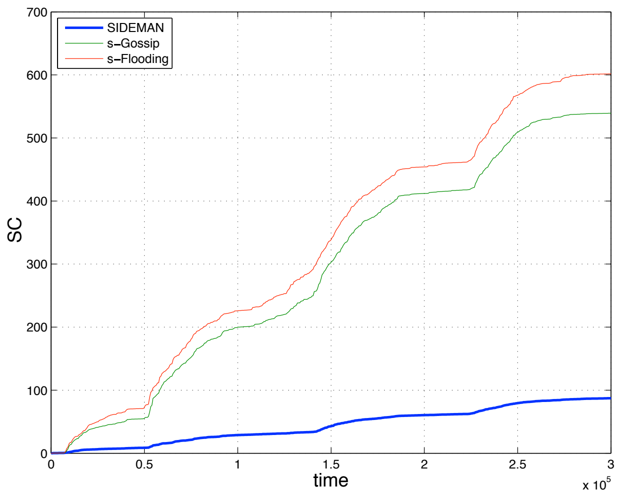


FIGURE 14
Service cache size in the Infocom 06 scenario.

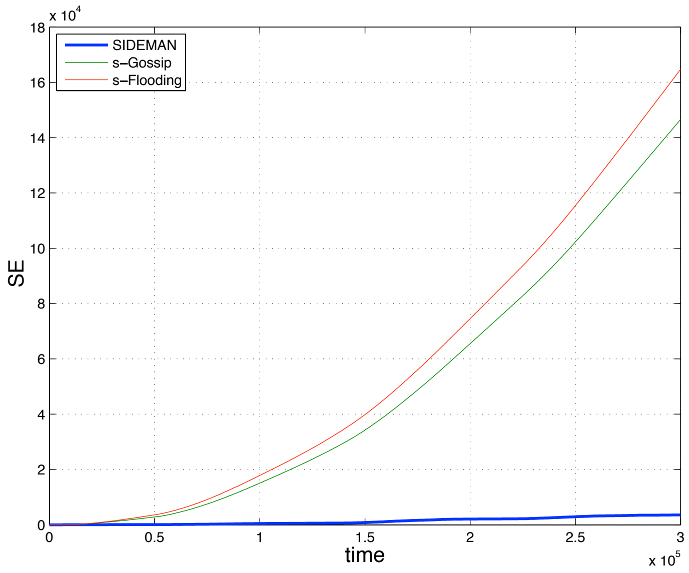


FIGURE 15
Number of services exchanged in the Infocom 06 scenario.

other nodes and as a consequence to exchange and store more services. In particular, in the Infocom 06 scenario nodes meets with the 91% of the other nodes, while nodes in the HCMM scenario meet only 65%.

Services Exchanged. The average number of services exchanged is shown in Figure 15. The number of exchanges grows in time and, and by the end of the observed network operation, nodes running s-Flooding and s-Gossip exchange the highest number of services. Nodes in the Infocom 06 scenario exchange more services than those in the HCMM one. In particular, nodes running s-Flooding and s-Gossip exchange respectively 16.6 and 14.5 times the total number of services available (10^4), while nodes running SIDEMAN exchange an average of 3200 services, i.e., only 32% of the services available.

Network Overhead. The network overhead of the three algorithms in shown in Figure 16. The number of packets exchanged grows similarly to what observed in the HCMM scenario. In particular, s-Flooding and s-Gossip incur the highest network overhead, while SIDEMAN obtains the lowest network overhead. Since in the Infocom 06 scenario nodes tend to encounter more nodes that in the HCMM scenario, the number of services and queries exchanged is higher, which explains the higher network overhead in this case. By the end of the simulation time nodes running SIDEMAN incur a network overhead 77.88% lower than that of s-Flooding and 75.71% lower than that of s-Gossip.

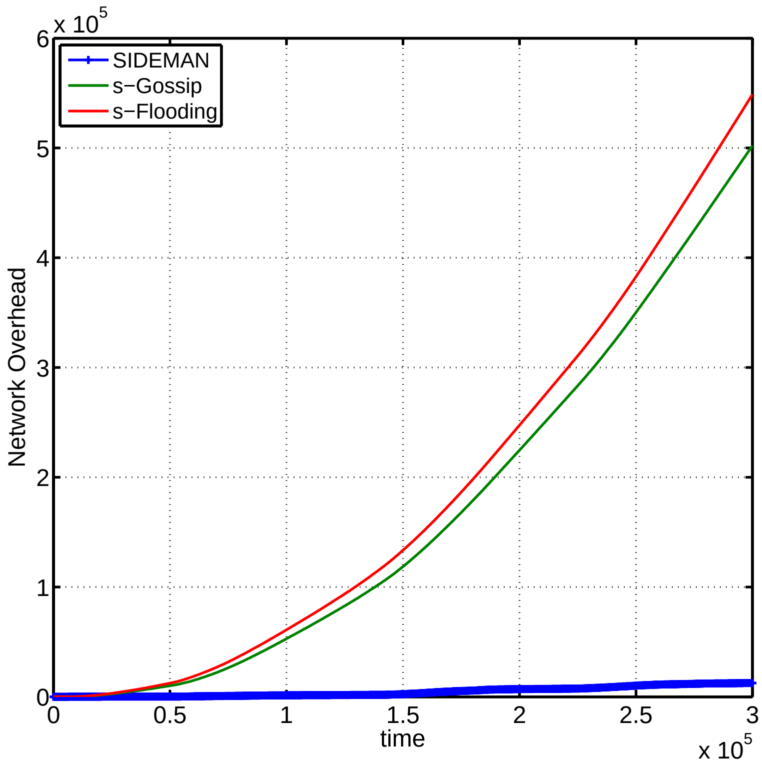


FIGURE 16
Network overhead in the Infocom 06 scenario.

7 CONCLUSIONS AND FUTURE WORK

In this paper we presented SIDEMAN, a service discovery algorithm for the users of a mobile social network. Differently from previous solutions, SIDEMAN exploits knowledge about the existing communities in the MSN to restrict the propagation of messages concerning services to the nodes that are most likely interested to them. In this way, it reduces the overhead and the energy consumption of the nodes due to query and service transmissions. The performance of SIDEMAN has been evaluated through simulations in synthetic scenarios based on a mobility model for MSN (HCMM) and in a scenario based on traces collected at the IEEE conference Infocom 2006. In particular, we have compared our algorithm with s-Flooding and s-Gossip, the social version of two popular data dissemination techniques. Metrics of interest include Recall (measuring how proactive a algorithm is in distributing services of interest), Gain (finding services in cache when needed), Energy

Cost, the time needed to reply to a service query, the size of the service cache and the number of services exchanged. Our results show that in both HCMM and Infocom 06 scenarios SIDEMAN is remarkably effective in obtaining flawless Recall, a Gain that is always comparable to that of s-Flooding and s-Gossip, providing most services to a node in reasonable time and at an incredibly lower energy cost than that required by the two other algorithms. Moreover, SIDEMAN effectively controls the size of the service cache and the number of services exchanged as well as the network overhead. The most noticeable achievement of SIDEMAN is obtaining values of Gain that are always comparable to those of s-Flooding and s-Gossip, incurring a significantly smaller network overhead and energy consumption. This suggests that SIDEMAN is a valuable algorithm for application scenarios in which nodes are resource constrained.

We plan to extend SIDEMAN by allowing nodes to collaborate for the diffusion of queries and services requested by other nodes. For instance, suppose Alice asks Bob for an information. If Bob has an answer, then he replies immediately, as currently with SIDEMAN. Otherwise, Bob starts asking to his own friends about it, and reports the information back to Alice at a later time. The Gain and, especially, the Query Response Time, are expected to be beneficially affected by this collaborative behavior of the nodes. We will also investigate data off-loading techniques to move part of the computation performed by a node to a cloud-based service to measure possible gains in performance, especially on the energy cost. In particular, since community detection can have quite a toll on the performance of service discovery, community detection could be delegated to a cloud-based service instead of being executing locally, with the added benefit that the cloud service could use centralized community detection algorithms, notoriously more effective and faster than distributed ones.

ACKNOWLEDGMENTS

This work was supported in part by the European Commission through the FP7 project universAAL (contract n. 247950) and CIP ICT PSP project ReAAL (contract n. 325189).

REFERENCES

[1] Unai Aguilera and Diego López-de Ipiña. (2012). A parameter-based service discovery protocol for mobile ad-hoc networks. In Xiang-Yang Li, Symeon Papavassiliou, and Stefan Ruehrup, editors, *Ad-hoc, Mobile, and Wireless Networks*, volume 7363 of *Lecture Notes in Computer Science*, pages 274–287. Springer Berlin Heidelberg.

[2] Apple, Inc., (2007). Bonjour technology white paper.

[3] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, editors. (March 5 2013). *Mobile Ad Hoc Networking: Cutting Edge Directions*. IEEE Series on Digital & Mobile Communication. IEEE Press and John Wiley & Sons, Inc., Piscataway, NJ and Hoboken, NJ, second edition.

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

[5] C. Boldrini, M. Conti, and A. Passarella. (October 27–31 2008). Contentplace: Social-aware data dissemination in opportunistic networks. In *Proceedings of ACM MSWiM 2008*, pages 203–210.

[6] C. Boldrini, M. Conti, and A. Passarella. (June 23–26 2008). Context and resource awareness in opportunistic network data dissemination. In *Proceedings of IEEE WoWMoM 2008*, pages 1–6, Newport Beach, CA.

[7] C. Boldrini and A. Passarella. (2010). HCMM: Modelling spatial and temporal properties of human mobility driven by users social relationships. *Computer Communications*, 33(9):1056–1074.

[8] E. Borgia, M. Conti, and A. Passarella. (June 12–15 2011). Autonomic detection of dynamic social communities in opportunistic networks. In *Proceedings of IEEE Med-Hoc-Net 2011*, pages 142–149, Favignana, Italy.

[9] C. Campo, M. Munoz, J. C. Perea, A. Marin, and C. Garcia-Rubio. (2005). PDP and GSDL: A new service discovery middleware to support spontaneous interactions in pervasive systems. In *Proceedings of the IEEE Percom Workshops 2005*, pages 178–182.

[10] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. (February 2006). Toward distributed service discovery in pervasive computing environments. *IEEE Transactions on Mobile Computing*, 5(2):97–112.

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

[12] M. De Domenico, A. Lima, and M. Musolesi. (2013). Interdependence and predictability of human mobility and social interactions. *IEEE Pervasive and Mobile Computing*, 9(6):798–807.

[13] Josep Díaz, Alberto Marchetti-Spaccamela, Dieter Mitsche, Paolo Santi, and Julinda Stefa. (2011). Socially-aware forwarding improves routing performance in pocket switched networks. In *Proceedings of ESA 2011*, pages 723–735.

[14] Vijay Erramilli, Mark Crovella, Augustin Chaintreau, and Christophe Diot. (2008). Delegation forwarding. In *Proceedings of ACM MobiHoc 2008*, pages 251–260.

[15] K. Fall. (August 25–29 2003). A delay-tolerant network architecture for challenged internets. In *Proceedings of ACM SIGCOMM 2003*, pages 27–34.

[16] A. Ferscha. (January 2012). 20 years past Weiser: What's next? *IEEE Pervasive Computing*, 11(1):52–61.

[17] M. Girolami, P. Barsocchi, S. Chessa, and F. Furfari. (June 24–26 2013). A social-based service discovery protocol for mobile ad hoc networks. In *Proceedings of IEEE Med-Hoc-Net 2013*, pages 53–60.

[18] E. Guttman, (1999). Service location protocol version 2. IETF RFC 2608.

[19] P. Hui, J. Crowcroft, and E. Yoneki. (November 2011). BUBBLE Rap: Social-based forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 10(11):1576–1589.

[20] Pan Hui, Eiko Yoneki, Shu Yan Chan, and Jon Crowcroft. (2007). Distributed community detection in delay tolerant networks. In *Proceedings of ACM/IEEE MobiArch 2007*, pages 7:1–7:8.

[21] R. Jathar, V. Yadav, and A. Gupta. (2014). Using periodic contacts for efficient routing in delay tolerant networks. *Ad-Hoc and Sensor Wireless Networks*, 21(3-4):283–308. cited By 0.

[22] Wei jen Hsu, Thrasyvoulos Spyropoulos, K. Psounis, and A. Helmy. (May 2007). Modeling time-variant user mobility in wireless mobile networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 758–766.

[23] S.-S. Kang. (2012). Pervasive peer to peer content distribution in ad hoc networks. *Ad-Hoc and Sensor Wireless Networks*, 14(1-2):9–25. cited By 0.

[24] Nicolas Le Sommer and Yves Mahéo. (November 2011). Olfserv: an opportunistic and location-aware forwarding protocol for service delivery in disconnected manets. pages 115–122.

[25] Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. (2006). BAR gossip. In *Proceedings of USENIX OSDI 2006*, pages 14–14.

[26] H Lim and C Kim. (2001). Flooding in wireless ad hoc networks. *Computer Communications*, 24(34):353 – 363.

[27] K. C.-J. Lin, W.-T. Lin, and C.-F. Chou. (2011). Social-based content diffusion in pocket switched networks. *IEEE Transactions on Vehicular Technology*, 60(9):4539–4548.

[28] Cong Liu and Jie Wu. (2008). Routing in a cyclic mobispace. In *Proceedings of ACM MobiHoc 2008*, pages 351–360.

[29] L. Liu, J. Xu, N. Antonopoulos, J. Li, and K. Wu. (2012). Adaptive service discovery on service-oriented and spontaneous sensor systems. *Ad-Hoc and Sensor Wireless Networks*, 14(1-2):107–132. cited By 7.

[30] J. Lloret, L. Shu, R. Lacuesta, and M. Chen. (2012). User-oriented and service-oriented spontaneous ad hoc and sensor wireless networks. *Ad-Hoc and Sensor Wireless Networks*, 14(1-2):1–8. cited By 6.

[31] Miller McPherson, Lynn S. Lovin, and James M. Cook. (2001). Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27(1):415–444.

[32] A. Mei, G. Morabito, P. Santi, and J. Stefa. (April 10–15 2011). Social-aware stateless forwarding in pocket switched networks. In *Proceedings of IEEE INFOCOM 2011*, pages 251–255.

[33] Microsoft Corporation, (2008). Universal plug and play: Background. http://www. upnp-hacks.org/upnp.html.

[34] T. D. Nguyen and S. Rouvrais. (2007). A socially inspired peer-to-peer resource discovery service for delay tolerant networks. In *Proceedings of OTM 2007*, volume 2, pages 960–969.

[35] D. Oberle, A. Barros, U. Kylau, and S. Heinzl. (March 2013). A unified description language for human-to-automated services. *Information Systems*, 38(1):155–181.

[36] Elena Pagani, Lorenzo Valerio, and Gian Paolo Rossi. (2015). Weak social ties improve content delivery in behavior-aware opportunistic networks. *Ad Hoc Networks*, 25, Part B(0):314–329. New Research Challenges in Mobile, Opportunistic and Delay-Tolerant Networks Energy-Aware Data Centers: Architecture, Infrastructure, and Communication.

[37] A.-K. Pietilainen and C. Diot. (April –19–25 2009). Social pocket switched networks. In *Proceedings of IEEE Infocom Workshops 2009*, pages 1–2, Rio de Janeiro, Brazil.

[38] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, Seong Joon Kim, and Song Chong. (June 2011). On the levy-walk nature of human mobility. *IEEE/ACM Trans. Netw.*, 19(3):630–643.

[39] J. Silvis, D. Niemeier, and R. D'Souza. (2006). Social Networks and Travel Behavior: Report From an Integrated Travel Diary. In *11th International Conference on Travel Behavior Research*.

[40] Nor Hashimah Sulaiman and M. Daud. (2012). A Jaccard-based similarity measure for soft sets. In *Proceedings of IEEE SHUSER 2012*, pages 659–663.

[41] A. B. O. Sullivan, R. W. Sheifler, J. Waldo, and A. Wollrath. The jini specification. Sun Microsystems, Inc.

[42] S. Tarkoma, C. Esteve Rothenberg, and E. Lagerspetz. (2012). Theory and practice of Bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, pages 131–155.

[43] N. Vastardis and K. Yang. (Third quarter 2013). Mobile social networks: Architectures, social properties, and key research challenges. *IEEE Communications Surveys & Tutorials*, 15(3):1355–1371.

[44] C. N. Ververidis and G. C. Polyzos. (Third quarter 2008). Service discovery for mobile ad hoc networks: A survey of issues and techniques. *IEEE Communications Surveys & Tutorials*, 10(3):30–45.

[45] Z. Wang, M. A. Nascimento, and M. H. MacGregor. (2013). Discovering periodic patterns of nodal encounters in mobile networks. *IEEE Pervasive and Mobile Computing*, 9(6):892–912.

[46] Zijian Wang, Eyuphan Bulut, and KS Boleslaw. (2010). Service discovery for delay tolerant networks. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 136–141. IEEE.

[47] M. Weiser. (July 1999). The computer for the 21st century. *SIGMOBILE Mobile Computing and Communication Review*, 3(3):3–11.

[48] X. Wu, K. N. Brown, and C. J. Sreenan. (2013). Analysis of smartphone user mobility traces for opportunistic data collection in wireless sensor networks. *IEEE Pervasive and Mobile Computing*, 9(6):881–891.

[49] S. Yang, Z. Li, M. Stojmenovic, C. Wang, and C. Jiang. (2015). Erupt: A role-based neighbor discovery protocol for mobile social applications. *Ad-Hoc and Sensor Wireless Networks*, 24(3-4):265–281. cited By 0.

[50] Eiko Yoneki, Pan Hui, ShuYan Chan, and Jon Crowcroft. (2007). A socio-aware overlay for publish/subscribe communication in delay tolerant networks. In *Proceedings of the 10th ACM Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, MSWiM '07, pages 225–234, New York, NY, USA. ACM.