Service Discovery in Mobile Social Networks

Michele Girolami and Stefano Chessa
ISTI–CNR, Pisa, Italy and
Department of Computer Science, Università di Pisa
Pisa, Italy
Email: {girolami, ste}@di.unipi.it

Stefano Basagni
ECE Department
Northeastern University
Boston, MA U.S.A.
Email: basagni@ece.neu.edu

Francesco Furfari
ISTI–CNR
Pisa, Italy
Email: furfari@isti.cnr.it

***Abstract*—We present a new service discovery algorithm, termed SIDEMAN, which considers human mobility for service dissemination and discovery. SIDEMAN takes advantage of mobile social networking characteristics, such as user membership to a restricted number of communities and interest for similar services among users in the same community. We evaluated the performance of SIDEMAN via simulations in a scenario based on traces collected at the IEEE conference Infocom in 2006. Our algorithm has been compared to the social version of two popular data dissemination techniques, namely, flooding and gossiping. We have measured how proactive an algorithm is in distributing services of interest (Recall), how many services are already with a user when they are needed (Gain), the energy cost for service discovery, and the time needed to reply a service query. We show that SIDEMAN obtains perfect 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 lower energy cost than that of the flooding and gossiping-based solutions.**

I. Introduction

A new paradigm of networking has recently emerged capitalizing on the opportunistic nature of human encounters to implement network services such as data forwarding, broadcast, routing, and service discovery. This new paradigm is often called *mobile social networking* to emphasize the human interaction on which it is based [1], or *pocket switching networking*, to indicate the kind of devices of which it is made [2]. The key ideas of mobile social networks (MSNs) is that the end user can actively participate in content distribution instead of just being a recipient of information from the telecommunication infrastructure. Studies on the mobility of humans and on their social relationships have therefore flourished with the intent of investigating how people, through the devices in their pockets, become actors in content distribution. For instance, for opportunistic data forwarding it becomes important to know how humans move in specific context, and whether there are common patterns in their movements [3].

In this paper we contribute to the problem of *disseminating and discovering services* in MSNs. With the term service, we refer to any kind of resource provided by a server accessible in the network. Examples include news, whether forecast, or information about traffic at a specific location. Early solutions concern infrastructure-based IP networks, and as such are not suitable for MSNs. Solutions for general mobile networks, such as ad hoc networks [4] and for pervasive communications [5] have been recently proposed. However, these solutions do not consider the all-human tendency to keep visiting familiar places and to aggregate with fellow people with overlapping interests, as typical of MSNs. We therefore propose a novel algorithm for service dissemination and discovery in MSNs. The algorithm, named SIDEMAN for *ServIce DiscovEry for Mobile sociAl Networks*, is social and opportunistic in that it takes advantage of human behavior concerning locations and community membership. It implements the two core operations of service discovery, namely, 1) *Service dissemination*, i.e., the distribution of services to the MSN users. This is realized by discovering and recognizing social communities and by a proactive diffusion of services among people with similar interests; 2) *Service query*, i.e., the process of requesting a missing service to a fellow user. This is implemented by a controlled query propagation mechanism aimed at avoiding extensive use of indiscriminate flooding.

The performance of SIDEMAN has been evaluated via simulations in a scenario where users move according to real traces collected at the IEEE Infocom 2006 conference site [6]. 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 [7], 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 [8] each user spreads 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 providing 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 for service discovery, and the average time needed to receive a response to a service query. Our results show that SIDEMAN outperforms both s-Flooding and s-Gossip. In particular, SIDEMAN achieves perfect Recall, being able to provide users with only the services they want, obtains a Gain that is comparable to that of the other protocols although exchanging a lower number of services, incurs a response time only slightly higher than that of s-Flooding, and spend a fraction of the energy required by service discovery with s-Flooding and s-Gossip.

The rest of the paper is organized as follows. Section II describes SIDEMAN. In Section III we evaluate the performance of SIDEMAN and compare it with that of s-Flooding and s-Gossip. Conclusions are provided in Section IV.

978-1-4799-4912-0/14/$31.00 ©2014 IEEE

II. The SIDEMAN Algorithm

Overview. SIDEMAN is an algorithm used by a node 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, and (ii) to proactively exchange services they might be interested in, so that a node has that service when it needs it.

While moving and encountering other nodes, node n_i keeps running a community detection algorithm for recognizing communities of nodes where it belongs. Whenever the node needs a service, it cheeks whether it has it in its service cache $\mathcal{A}_i$. If this is the case, it uses the service. Otherwise, it crafts a query q (a tuple of interests) and sends it to the members of its current community whose interests match those in q. In other words, the query q is broadcast only to those members of the community of n_i that share at least one interest of q (controlled flooding). If some of the nodes in the community have an answer to the query, they immediately reply to n_i and that service is stored in $\mathcal{A}_i$. If no answer is received for q, n_i adds it to its set of the pending queries PQ_i.

Every time a community is recognized, a node exchanges the services matching the community interests with the nodes in the community itself. The new received services are stored in $\mathcal{A}_i$. At this time, the node checks whether previous unanswered queries (stored in the set PQ_i) concern services now in the cache. All queries q answered by newly received services are removed from PQ_i. The queries left in PQ_i are then sent to the member of the community just detected. If some of the nodes in the community have an answers to the queries in PQ_i, they immediately reply to n_i that update $\mathcal{A}_i$ accordingly. An overview of the reactive and proactive components of SIDEMAN is shown in Fig. 1.

Algorithm description. The following algorithm describes how service discovery is performed at node n_i whenever it needs to use a service.

Algorithm 1 DiscoverService(s)

1: $C = \text{RecognizeCommunity}(T^i, N^i, CT^i, \tau)$
2: **if** $s \in \mathcal{A}_i$ **then**
3: use s
4: **else**
5: $q = SI_s$
6: $V_q = \{n_j \in C \mid \exists\ t_k \in q \text{ s.t. } t_k \in I_j\}$
7: CommunityBroadcast(q, V_q)

Node n_i starts by recognizing its current community C by executing the following algorithm.

Algorithm 2 RecognizeCommunity(T, N, CT, τ)

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

In order to recognize a community the algorithm uses T, the history of contacts between node n_i and other nodes, N, the current neighbors of n_i, CT, the list of communities visited by n_i so far, and a threshold τ used for determining how similar two communities are. For determining the community C where it resides, n_i executes a community detection algorithm A using the contact history T_t^i and the neighborhood N_t^i of node n_i at time t as input. (SIDEMAN is independent of the specific algorithm used to detect communities, and can use any of the detection algorithms proposed for MSNs [9], [10].) The detracted community C is then *recognized* if there is a community C' in CT that is *similar* to C according to a given similarity index. We use the Jaccard index with threshold τ [11]. If a community C' is found that is Jaccard-similar to C 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 community interests must be collected by asking them to every node in C (function $getInterests(\cdot)$ in line 5). Community C is then stored in CT (line 6). The recognized community C is finally returned (line 7). We stipulate that C encodes both the community (the set C of its nodes) and the community interests $\mathcal{I}_C$. We recognize communities through the Jaccard index for the purpose of using the function $getInterests(\cdot)$ as little as possible. This function entails a possibly large number of communications with neighboring nodes, thereby lowering performance, requiring longer times, and higher energy consumption. We observe that communities are recognized well over 50% of the times, i.e., the function $getInterests(\cdot)$ is called a negligible amount of times.

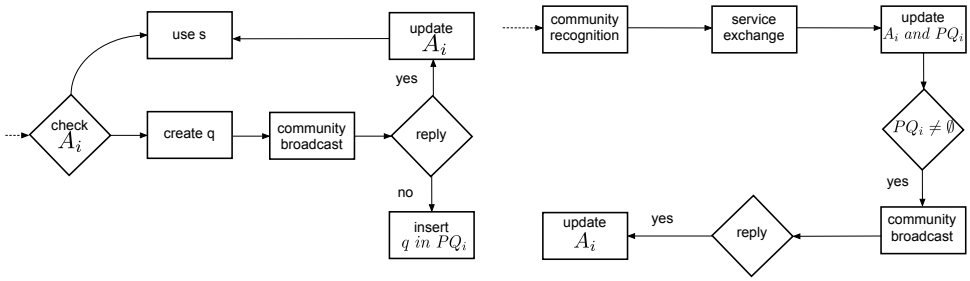
Once it has recognized a community C (line 1, Algorithm 1), node n_i checks whether it already has service s (line 2). In the positive, it uses it. Otherwise, it generates a query q for service s based on the service interests SI_s of s (line 5). Node n_i then computes the set $V_q \subseteq C$ of nodes that can potentially answer the 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 broadcast to nodes in V_q (line 7).

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

Algorithm 3 ForwardServicesAndQueries

1: $C = \text{RecognizeCommunity}(T^i, N^i, CT^i, \tau)$
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: CommunityBroadcast(S_t, V_t)
8: **for all** $q \in PQ_i$ **do**
9: **if** $t \in q$ **then**
10: CommunityBroadcast(q, V_t)

Node n_i starts with recognizing its community. To this purpose it uses Algorithm 2 that returns C and $\mathcal{I}_C$. For every interest $t \in \mathcal{I}_C$ node n_i performs the following actions. 1) It



(a) Reactive component.

(b) Proactive component.

Fig. 1: Overview of SIDEMAN.

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 member of C. 2) It broadcasts its services to nodes in its community sharing the same interests (line 7). 3) It selects those pending queries from PQ_i concerning t, and broadcasts them to members of its community sharing t (line 10).

Whether reactively sending out service queries (Algorithm 1) or proactively exchanging services and disseminating pending queries (Algorithm 3), after a community broadcast some of node n_i neighbors might transmit responses. Algorithm 4 describes how n_i reacts to receiving response m.

Algorithm 4 OnMessageReception(m)

1: **if** m is a query **then**
2: Extract from $\mathcal{A}_i$ the services $\{s_1^i, \ldots, s_k^i\}$ matching query m
3: Unicast($\{s_1^i, \ldots, s_k^i\}$, sender(m))
4: **else**
5: Update($\mathcal{A}_i$, m)

Upon receiving response m, node n_i checks whether it is a query or a service. If m is a query then it determines all its services that answer that query, if any. These services are sent directly (i.e., through a unicast transmission) to the node sender(m) that originated the query. If m is 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 so, the corresponding entry is updated. Otherwise, m is added to $\mathcal{A}_i$.

III. Performance Evaluation

We evaluate the performance of SIDEMAN through Java-based simulations that take into account characteristics of MSNs. We implemented SIDEMAN as well as s-Flooding and s-Gossip.

Simulation scenario. We consider a MSN scenario where nodes move in a given area according to real traces gathered from participants to the IEEE Infocom conference from April 24 to April 27 2006 [6]. We consider data of 78 conference attendants who 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 out a survey about her/his interests, nationality, language spoken, etc. (We use this survey to assign interests to the nodes).

Simulator and parameters. We implemented SIDEMAN in a home-grown Java-based simulator to evaluate its performance and compare it to that of other discovery algorithms. The simulator can run one or more discovery algorithms simultaneously and use different mobility models (e.g., Infocom 06 or synthetic traces). The simulator implements three important components of service discovery in MSNs: (i) Contact history maintenance, (ii) community detection and (iii) the service discovery algorithms. (i) Contact history maintenance concerns the contact history at each node. (ii) Community detection is implemented by a well known solution, called AD-SIMPLE [9]. 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 [10]. The reasons for this choice are multifold: First of all, AD-SIMPLE is a distributed algorithm, suitable to run in MSNs, and detects communities that are very similar to those detected by SIMPLE [9]. AD-SIMPLE also 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. Finally, (iii) service discovery is implemented by SIDEMAN (Section II). To demonstrate its effectiveness in discovering and advertising services, the performance of SIDEMAN is compared to that of s-Flooding and s-Gossip, which are the "social" version of traditional flooding [7] and gossiping [8]. 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.

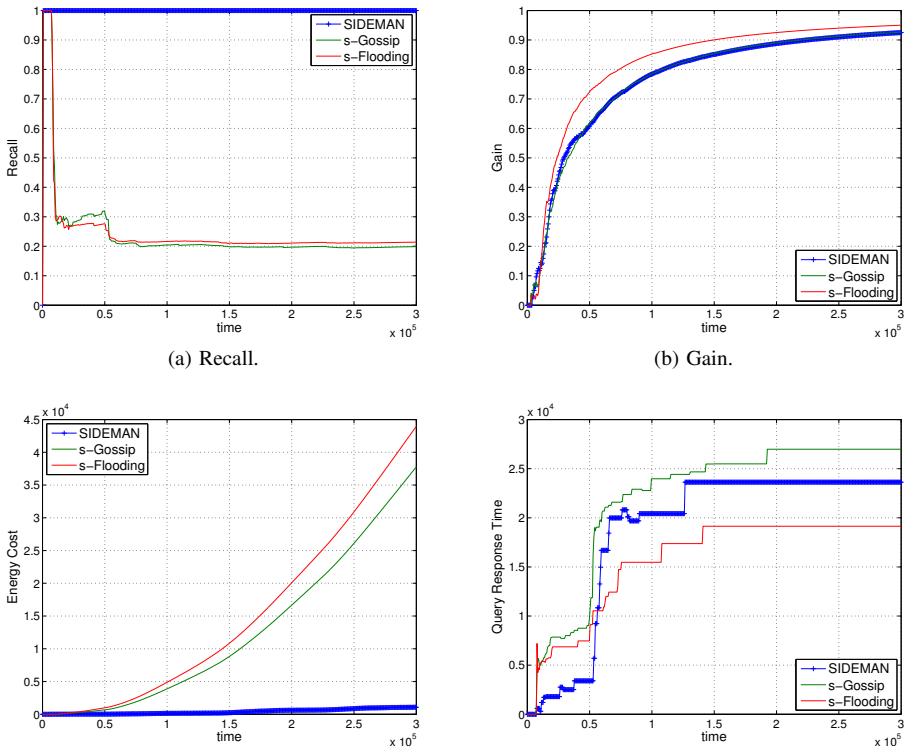
Node behavior is determined by query generation rate, service generation rate and the distribution of interests to nodes. The query generation rate determines how many queries are generated by the nodes. We modeled this rate as a Poisson process of intensity λ queries per second. In our simulations we set $\lambda = 3$. In particular, every second a number x of query is generated and assigned to x 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 y of services is generated and assigned to y nodes selected randomly and uniformly among all nodes. Each time the y services are drawn randomly and uniformly from a set of $m = 10000$ services. We use the association between interests and nodes that comes with the traces. Energy consumption is computed according to the data sheet of the WiFi/ Bluetooth chip Broadcom®BCM4330 of the Samsung Galaxy S III smart phone. Table I summarizes the simulation parameters.

TABLE I: Simulation parameters.

Transmission range	30m
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
Simulated time	about 201600s

Metrics. We compare the performance of SIDEMAN, s-Flooding and s-Gossip with respect to the following metrics.
1) *Recall* R, 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. Clearly, R is in $[0, 1]$. The lower the Recall the more a node stores uninteresting services, i.e., services that it will unlikely use. The higher the Recall the more a node stores services of interest, which are services that it will likely access. 2) *Gain* G, defined as the ratio between the number of times a node finds the service s in its service cache and the number of times it 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. The value of G is in $[0, 1]$.
3) *Energy Cost per node* (in Joules), 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. Its computation does not consider the community detection algorithm, thus indicating only the energy consumption of service discovery.
4) *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.

Results are shown as time progresses, to provide an indication of how nodes converge to a state in which most of the services they need are in their cache. Fig. 2 depicts the results of SIDEMAN, s-Flooding and s-Gossip.
Recall. Results concerning Recall are shown in Fig. 2a. 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 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. Nodes running s-Flooding and s-Gossip are more likely to exchange services in which they are not interested.
Gain. Results for the Gain are shown in Fig. 2b. The trend is the same for all algorithms. By the end of the observation time the Gain reaches the value 0.94. 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.
Energy Cost. Fig. 2c shows the energy cost incurred by each node while running the three algorithms. 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, we observe that devices running s-Flooding and s-Gossip deplete the 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. We also measured the overhead of the three protocols. At the end of the observation time, SIDEMAN network overhead is $96,44\%$ $(96,58\%)$ lower than that of s-Flooding (s-Gossip).
Query Response Time. Results for the query response time are shown in Fig. 2d. The time needed to respond to a query grows as time progresses. This is because nodes tend to visit the same communities and so it is unlikely that it will receive a 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, but we have observed that this happens with low probability. The query response time 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 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



(a) Recall.

(b) Gain.

(c) Energy cost per node.

(d) Query response time.

Fig. 2: Comparative performance evaluation of SIDEMAN, s-Flooding and s-Gossip.

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 the number of queries sent by nodes: 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.

IV. Conclusions

We presented SIDEMAN, a service discovery algorithm for users in a mobile social network allowing nodes belonging to the same community to exchange only services of interest. The performance of SIDEMAN has been evaluated through simulations in a scenario based on traces collected at the IEEE conference Infocom 2006. 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 and the time needed to reply to a service query. Results show that SIDEMAN is 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.

Acknowledgments

This work was supported in part by the European Commission through the FP7 project universAAL (contract n. 247950).

References

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

[2] A.-K. Pietilainen, C. Diot, Social pocket switched networks, in: Proceedings of IEEE Infocom Workshops 2009, Rio de Janeiro, Brazil, 2009, pp. 1–2.

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

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

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

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

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

[8] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, M. Dahlin, BAR gossip, in: Proceedings of OSDI 2006, 2006, pp. 14–14.

[9] E. Borgia, M. Conti, A. Passarella, Autonomic detection of dynamic social communities in opportunistic networks, in: Proceedings of IEEE Med-Hoc-Net 2011, Favignana, Italy, 2011, pp. 142–149.

[10] P. Hui, E. Yoneki, S. Y. Chan, J. Crowcroft, Distributed community detection in delay tolerant networks, in: Proceedings of ACM/IEEE MobiArch 2007, 2007, pp. 7:1–7:8.

[11] N. H. Sulaiman, M. Daud, A Jaccard-based similarity measure for soft sets, in: Proceedings of IEEE SHUSER 2012, 2012, pp. 659–663.