First publ. in: 1st IFIP Wireless Days" 2008, 24-27 november 2008, pp. 1-5

I Seek for Knowledge: Exploiting Social Properties in Mobile Ad Hoc Networks

Sebastian Kay Belle* Muhammad Arshad Islam* Marcel Waldvogel*
* University of Konstanz
Distributed Systems Laboratory
Konstanz – Germany
firstname.lastname@uni-konstanz.de

***Abstract*—New social networks are born each day, at a formal conference, at informal social gathering, at family reunions etc. Internet has already been playing an important role in modern socialising. But it is still not the optimal way of interaction as one has to be very active updating profiles. With the easy access to mobile devices, modern technologies have now started to adopt to new ways of socialising. mobile devices accompany their users almost all the time, they can record and observe their users behavior as well as gather information about their social circle. Therefore, they can help users to get information from contacts, that they potentially not even know. In this paper we put our efforts towards the initial design of an architecture that will sniff for information around the user's surrounding, leveraging useful answers on their demand.**

I. Introduction

Nowadays, we face a novel kind of social networking, *virtual social networks* as enforced by well known platforms like *Facebook*, *MySpace* or *Friendster* – just to name a few – that have more than 50 million registered users. This new approach to social networks somehow reflects the old habit of mankind that people tend to arrange themselves in groups with similar interests to share and exchange their knowledge.

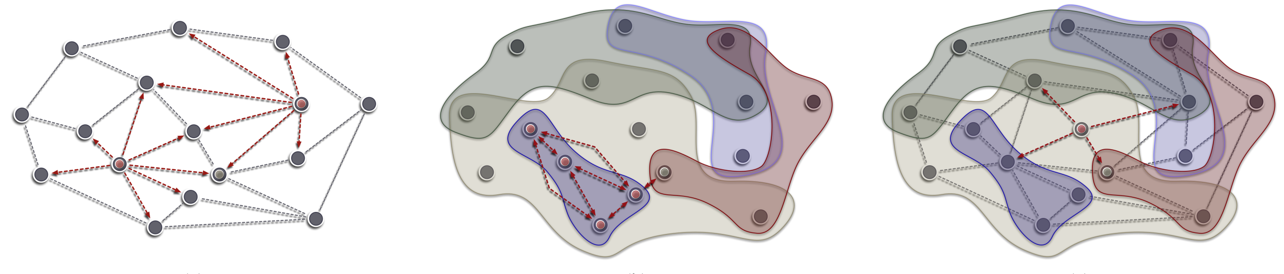
Another aspect of modern social networking is that information is no longer solely stored and found in databases or documents in the World Wide Web. Instead, information, or to be more precise, *knowledge* is shared by users worldwide. Hence, social networks become even more prominent when searching for information to solve a problem. Simplified, it all comes down to the well known adage:*"It's not what you know, it's who you know"*. Therefore, the question we face is less *how we find the information,* but more how we get in contact with *someone who has the expertise*. These *group-forming networks*, as discussed in Reed's law [13], have an utility that grows exponentially with the number of participants – unlike traditional networks whose utility merely grows linearly or quadratically. Anyway, note that the term "friend" gets somewhat altered in the context of this new type of social network. Relationships become *binary*, either you are connected or you are not connected. These social networks alter the type of relationship between people, personality becomes somewhat irrelevant, only the mere interest for a specific topic connects one person to another. Further, modern social networking bypasses the geographic context in which people reside. Technology simplifies it to stay in contact with others, even in remote locations of our planet, for instance by e-mail or chat. However, the most obvious device we use to stay in contact with others – and probably nowadays the most mobile devices can aid us to leverage the next level of social networks, *mobile social networks*. *personal* device – are mobile phones.

First, we define why it is important to rethink what we like to achieve with modern (mobile) social networks. Second, we state the difficulties to face when as we integrate these aspects in mobile social networks.

a) Why do we care?: (1) Connecting to experts that have some specific knowledge facilitates problem solving as these experts can provide quick access to answers we could not solve on our own, or at least, only with an high effort. (2) Modern social networks build upon a *pull* type approach where users explore the available connections in the network. However, a *push* type approach as we know it from *real-world social networks* – when we get introduced to others by colleagues of friends – would ease the use of such a system. (3) Related to the previous point is the third aspect that is specifically inherent to virtual social networks, *transitivity*. Transitivity is a powerful way to explore social connections by following links to n^{th} degree contacts. However, due to the *pull* approach, utilising transitivity is somewhat limited as the user is responsible to dig out experts and does not get any recommendations.

b) What problems to solve?: (1) Modelling an automatic recommendation system based on the *push* approach. (2) Modelling the transitive relationship in a system that facilitates the desired *push* approach. (3) Including social information (e.g. expertise in a specific topic) of the users in the social network.

In this paper we propose a system design based on mobile devices, building an implicit infrastructure for mobile social networks. The proposed system integrates social information of its users stored in Bloom filters [3] to facilitate the look-up of experts or people that share a common interest. We define a two-way message protocol that exploits the social network's inherent transitive connections, enhanced with the social information of the users in the network, to route messages to a suitable receiver. Further, this message protocol enables a receiver to reply to the initial sender, using a pointed multicasting approach, while maintaining the replier's privacy



(a) (b) (c)

Fig. 1: Simplified illustration of the concept of centrality measurements fig.(a), semantic group models fig.(b), and the enhancement of connections in a social network by exploiting social information about the nodes fig.(c).

as well as the privacy of the initial sender.

II. Related Work

Following, we give an overview of related work in the area of *Mobile Ad Hoc Networks* (MANETs) and *Delay Tolerant Networks* (DTNs). We focus on research that exploits social information in these networks. Further, we review some basics on Bloom filters used later on in section III.

1) Exploiting Social & Semantic Information: In social sciences, researchers have been debating about different parameters to classify the satus of someone in a social network. These parameters include the number of contacts, the closeness to contacts, and location in the network. However, the concept of *Betweenness Centrality* [6] has gotten the most attention (c.f. Fig1(a)). Using *Betweenes Centrality* those nodes in the network are addressed for routing that have the highest number of connections to other nodes in the network. Daly and Haahr [4] propose a metric to identify characteristic nodes in a delay tolerant MANET that can serve as bridge nodes to pass messages to the intended receiver. To identify these nodes, [4] exploit social analysis techniques as a measurement of centrality in a MANET to enhance routing. Their idea is derived from the characteristics of the *small world phenomenon* which states that individuals are often linked by short links in overlapping circles of acquaintances [11]. Miklas et al. [10] exploit the use of social information in mobile systems, conducting a simulation on the "Reality Mining" dataset of the MIT Media Lab [1]. In their simulation they classify users into *strangers* and *friends* due to their number of encounters. This simulation shows that this simple classification already yields a gain in the message delivery performance in DTNs in terms of the time a message needs to arrive at its destination. Zhao et al. [15] introduce semantic models for efficient routing in DTNs by defining group membership models (*Temporal Membership*, *Temporal Delivery*, and *Current Membership*) for routing messages through the network (c.f. Fig.1(b)). These models define membership groups that could be interested in receiving a message depending on the location and/or time. A similar semantic model was proposed by Gong et al. [7].

In contrast to the approaches of [15] and [7] we follow the idea of [10] to utilise social information. However, we do not only classify users into *friends* and *strangers* but add a third class to the system, *trusted friends*. Additionally, we do not stop at this classification in our design. We also introduce transitivity in the social information of a user, simplified, we combine *Betweeness Centrality* and *Group Membership Models* to identify users that are either experts in a specific topic, or users that are connected to experts, respectively (c.f. 1(c)).

2) Bloom Filters in Networking: Bloom filters [3] are simple space-efficient randomized data structures that maintain a set of elements and support membership queries while allowing false positives. Formally, a Bloom filter is a bit array of length m that utilises k hash functions to map a set of n elements, with $n < m$, into the bit array. Each element is hashed to k distinct indices in the array setting the k indexed bits to 1. If a bit is set more than once by different elements the bit simply retains its value. The probability p_{err} of a false positive depends on the three parameters m, n, k of the Bloom filter and is calculated as::

$$p_{err} = (1 - e^{-kn/m})^k \tag{1}$$

Given a fixed size m for a Bloom filter as well as the intended (maximal) number of elements n, the optimal number of hash functions to uses can be obtained as[1]::

$$k = \ln 2 \cdot (m/n) \tag{2}$$

Bloom filters have been used to solve a variety of networking problems. Different variations of Bloom filters have been proposed in the last years, most of them suited towards a specific application area. Mitzenmacher [12] introduced the concept of compressed Bloom filters. He has shown that the size of a Bloom filter can be reduced without increasing p_{err} by introducing some modifications to the original concept. Guo et al. [8] proposed dynamic Bloom filters that can grow as needed by adding static sized Bloom filters to a list of Bloom filters as soon as the actual Bloom filter's false positive rate increases over a given threshold. Bauer et al. [2] propose simple boolean set operations on Bloom filters to combine sub-queries for efficient queries on distributed hash tables. However, Bloom filters are by far not the only way to go. Hurley and Waldvogel [9] have shown that Bloom filters can be outperformed by other techniques in the case that false negatives are admissible.

[1]Note that in practice k is rounded down to the next integer as this, in general does not increase p_{err} dramatically and yields better performance.

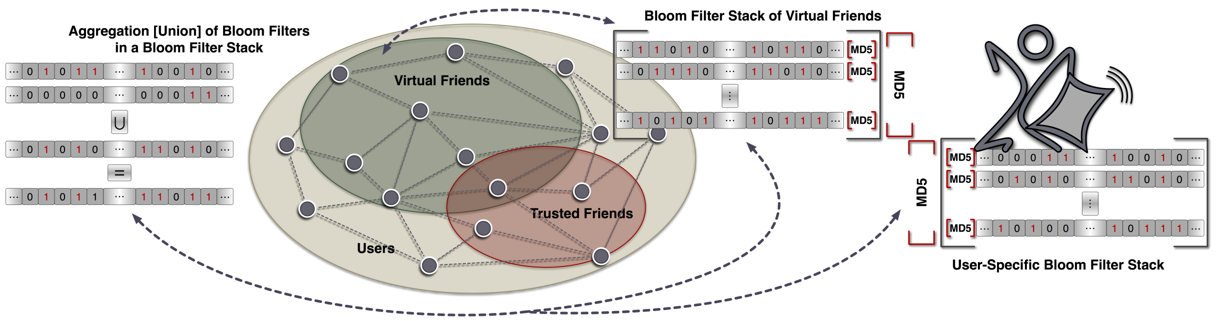

Aggregation [Union] of Bloom Filters in a Bloom Filter Stack
Bloom Filter Stack of Virtual Friends
Virtual Friends
Trusted Friends
Users
MD5
User-Specific Bloom Filter Stack

Fig. 2: Illustration of a user's profile. A user combines his profile (Bloom filter stack) with the profiles of his *virtual friends*. Other users in the system receive the aggregated profile as an approximation of the expertise the user has.

We use Bloom filters to encode the social information of users as they allow efficient set operations according to [2]. As already stated, we enhance our system with this user specific information and utilise this information in the look-up procedure of potentially interesting contacts as well as in our two-way message protocol. Therefore we can tolerate false positives, whereas false negatives are not preferable.

III. Mergenet Architecture

In the following section we present the design of the *Mergenet* architecture. We assume that users of our system are in possession of mobile devices capable of P2P communication. We aim *Mergenet* to be flexible enough to let users establish P2P connections through arbitrary techniques like WAN, LAN, Wifi, or Bluetooth. An user's profile is present on the mobile device giving the responsibility to the user to keep his profile up-to-date, especially if he uses multiple devices. Details about profiles will be presented in section III-A.

Mergenet takes into account the disparity among the users with respect to their social activities. Some users may like to be very social by introducing themselves to as many people as possible. On the other extreme, some may want to keep a very limited activity by maintaining a small number of close contacts. *Mergenet* classifies contacts as *users* and *virtual friends* similar to the idea of [10] depending on the frequency two devices see each other. However, in contrast to [10], we let users add *trusted friends* manually, or, depending on recommendations of the system, let an user choose to add a virtual friend in his list of trusted contacts. For a user p the set of virtual friends C_{VF} is defined according to a threshold α_p as::

$$C_{VF} = \{c \mid c \in C_U,\ T_c \geq \alpha_p\}, \tag{3}$$

where C_U is the set of all the users p encountered, $T_c = M_c/(\sum_{i \in C_U} M_i)$, M_c is the meeting count of p with $c \in C_U$, and M_i is the meeting count of p with $i \in C_U$. The threshold α_p varies for all users and depends on the embedding of p in the social network. At the moment we are running experiments to identify the best matching α_p for a user p.

Following, we present two definitions we will use throughout the next sections::

- *Client* – an user interested in either introducing himself to the community or to enquire some specific information.
- *Responder* – an user interested in getting to know a new *client* or to respond to a query from a client.

A. Profile Structure

We propose a profile structure based on Bloom filters. Every profile contains a set of Bloom filters $B_j,\ j \in \mathbb{N}_1$ in a stack $BS_i = \{B_1^i, B_2^i, \ldots, B_j^i\},\ i \in \mathbb{N}_1$. BS_1 denotes the Bloom filter stack for the current user while $i > 1$ denote the Bloom filter stacks for the users that came in direct contact with the current user. The first Bloom filter B_1^i in each stack is considered mandatory and contains the minimum information for the profile to exist. B_1^i will be populated with at least the identification information of the users, while the remaining Bloom filters correspond to an user's interests (e.g. music, movies, literature etc.). For every Bloom filter in the stack we compute the check sum. Whenever the profile is updated, the changes will be hashed in the corresponding Bloom filter and the check sum is recomputed. Thus, other users in the network perceive the change of a contacts profile. Formally, let $CS^{B_j^i} = f^B(B_j^i)$ be the check sum for the Bloom filter B_j^i. Then, the check sum of BS_i would be computed as $CS^{BS_i} = f^{BS_i}(CS^{B_k^i}),\ k = 1, \ldots j$. This helps to shortlist the number of Bloom filters in the case of profile exchange.

Whenever two users come in contact with each other for the first time they create a new Bloom filter stack for each other, incrementing their corresponding values of i. At the time of profile exchange, *Mergenet* computes the union of all Bloom filters at one level of the stack before transmitting them. Formally, let $BS^T = \{B_2^T, \ldots, B_j^T\}$ be the Bloom filter stack to transmit. BS^T is computed as follows, omitting the mandatory Bloom filter::

$$BS^T = \forall BS_i : \bigcup_j B_j,\ j > 1. \tag{4}$$

Thereby, we can ensure the transitivity property in the network. I.e., if user A shares her profile with B, and B shares her profile with C, C's profile also contains information about A without being connected to A directly. However, *Mergenet* permits the aggregation of profiles only for Bloom filter stacks $BS_i,\ i > 1$ of *virtual friends*. Bloom filter stacks associated with users $u \notin C_{VF}$ will not be aggregated into BS^T. These Bloom filter stacks will be kept separate, as they relate to

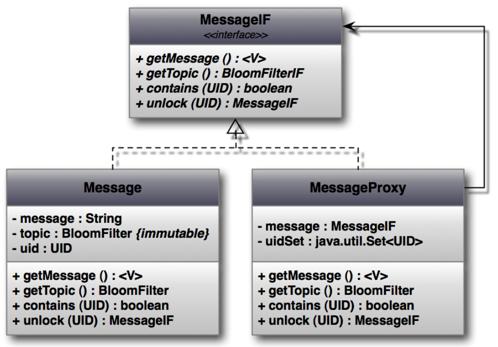

MessageIF
<<interface>>
+ getMessage () : <V>
+ getTopic () : BloomFilterIF
+ contains (UID) : boolean
+ unlock (UID) : MessageIF
Message
- message : String
- topic : BloomFilter {immutable}
- uid : UID
+ getMessage () : <V>
+ getTopic () : BloomFilter
+ contains (UID) : boolean
+ unlock (UID) : MessageIF
MessageProxy
- message : MessageIF
- uidSet : java.util.Set<UID>
+ getMessage () : <V>
+ getTopic () : BloomFilter
+ contains (UID) : boolean
+ unlock (UID) : MessageIF

Fig. 3: The UML class structure of the message design.

users of the system that are not encountered frequently. Fig. 2 illustrates the relationship between an user's profile encoded in a Bloom filter stack, the Bloom filter stacks of trusted friends, and the aggregation of an user's profile and the profiles of her *virtual friends*. Note that the set of *trusted friends* is not necessarily a subset of *virtual friends*.

Due to the aggregation of the different Bloom filters in the Bloom filter stacks, the false positive rate p_{err} (c.f. Eq. (1)) can increase dramatically. In our design we focus on three different approaches to overcome this problem. (1) Use large Bloom filters with a fixed size $m = 2^{31}$ that can be compressed before transmission as proposed by [12]. (2) Use *Dynamic Bloom Filters* (DBF) [8]. Thereby, the size of any Bloom filter $B_j \in BS_i$ will be kept flexible and grows dynamically as needed. Whenever the existing Bloom filter gets heavy to sustain the false positive rate, another Bloom filter of equal size is created while the old one is sent to storage and treated as legacy Bloom filter. This enables our system to transmit only chunks of the profile that were updated instead of always transmitting a large Bloom filter. Furthermore, clients with low populated profiles will not be burdened by unfilled Bloom filters. (3) Use *Attenuated Bloom Filters* [14] that only aggregate the information of user profiles up to a predefined depth of transitive links. Actually, this is the most promising approach so far.

B. Protocol definition

The transmission protocol consists of two kinds of exchange – *profile exchange* and *information exchange*.

1) Profile Exchange: Mergenet identifies every user by an unique identifier (UID). Whenever two users encounter each other one user assumes the role of the *Client*, the other as the *Responder*. The possible deadlock in this situation can be removed by attempting random waiting and reattempting to acquire the role. If one device acquires the role of the *Client* c, it connects to the *Responder* r. Following, the profile exchange is achieved as shown in Alg. 1. Method calls on the *Client* side are denoted as $c.methodCall()$, methods executed at the *Responder* side are written as $r.methodCall()$. This syntax will be used throughout the presented algorithms.

2) Information Exchange: Information exchange includes any kind of queries users may ask among each other. *Mergenet* retrieves the suitable *Responder(s)* to a query by directing

Algorithm 1 Profile Exchange & Update

```
1: $c.send(UID, CS^{BS_c^T})$ {//send initial information to $r$}
2: **if** $r.isNew(UID)$ **then**
3:     $r.add(UID)$
4:     $BS_c^T \leftarrow c.exchangeBS(BS_r^T)$ {//exchange stacks}
5:     $r.addBFStack(UID, CS^{BS_c^T}, BS_c^T)$
6: **else if** $r.isUpdated(UID, CS^{BS_c^T})$ **then**
7:     $BS_c^T \leftarrow c.updateBS()$ {//update stack of $c$}
8:     $r.replaceBFStack(UID, CS^{BS_c^T}, BS_c^T)$
9: **end if**
```

the query to only those users who have knowledge in the area the query belongs to. To route the message to a suitable *Responder*, *Mergenet* includes the topic of the query in the message as a Bloom filter that contains the relevant bits to identify the topic. Note that every query has a lifetime limited by t_{exp}, its *expiry time*.

Mergenet assumes that c maintains a list of all alive queries $Q = \{q_0, \ldots, q_k\}$ sorted by their expiry time. A query q is an object of type `MessageIF` (c.f. III-B2a & Fig.3). As a precondition for information exchange, *Mergenet* assumes that user profiles are up-to-date according to Alg.1.

a) Message Design: In *Mergenet* we utilise a proxy enveloping technique. Using this technique, the *Responder* will only "see" the set of forwarding users of the query (including the initial *Client*). The *Client* will only see the answer to his query but not the person who gave the answer. This technique preserves the privacy of the *Client* as well as the privacy of the *Responder*. We design this enveloping technique by utilising the *Decorator* design pattern [5] as depicted in Fig.3. The `MessageIF` exposes four methods to retrieve the topic of the message encoded in a Bloom filter, the message text, a method to check if a given UID is included in the set of forwarding nodes of the message, and a method to unlock the method up to an envelope created by the node with the corresponding UID. The initial *Client* creates the `Message` object adding the topic of the message, the message itself, and her UID. Thereafter, the message is wrapped in a `MessageProxy` object, adding an arbitrary number of optional message receivers from her from her list of *virtual friends*. These additional nodes are later used for passing back the response to a query.

b) Query & Response Forwarding: Forwarding a query q to a suitable *Responder*, or routing the answer to a query back to its initial *Client*, involves the steps described in Alg.2. According to the previous definition, we write $q.messageCall()$ to represent a method call of the message object. Simplified, the current *Client* – either the initial creator of the query or an intermediate node – passes the message to the next *Responder* that is identified as an user that can either answer the query, or knows another person that can answer the query, respectively. Whenever c connects to a *Responder* r it checks if r is either an expert for the topic of any message $q \in Q$, or can serve as a forwarding node for the message. Therefor, c and r first exchange or update their profiles according to Alg.1. As r exposes his aggregated profile BS_r, the profile

Algorithm 2 Query & Response Forwarding

```
1: for all $q \in c.Q$ do
2:   if $q.isQuery()$ then
3:     if $r.isExpert(q.getTopic())$ then
4:       $r.handleQuery(q)$ {// c.f. Alg. 3}
5:     end if
6:   else
7:     if $q.contains(r.UID)$ then
8:       $temp \leftarrow r$
9:       $r \leftarrow c$
10:      $c \leftarrow temp$
11:      $c.handleQuery(q.unlock())$ {// c.f. Alg. 3}
12:    end if
13:  end if
14: end for
```

information of his n^{th} degree contacts will be included as well. Thus, c does not actually know if r can answer the query or if r simply serves as an intermediate node. This forwarding and back-routing scheme is further illustrated in Fig.4. The thick red edges correspond to a query-forwarding from the initial *Client* on the left to the final *Responder* on the right. Thereby, the query is forwarded by the intermediate users, represented by the (red) nodes in the centerline of the figure. Each of the intermediate nodes encapsulate the message in another `MessageProxy` object (the message envelopes), adding an arbitrary number of further contacts that are identified as *virtual friends* of the active node (c.f. Eq. (3)). These additional nodes serve as proxies that can be used later for routing back the answer of the query to the initiating *Client*. The grey areas in the back of the figure correspond to the envelopes an intermediate node creates before forwarding the message. Every message envelope contains the information of the creating node, including the additional proxy nodes, and each envelope, including all envelopes created by nodes after the actual node, can be unlocked by the creating node.

During response forwarding the final *Responder* keeps the message as long as the query is alive and passes it to every node, including proxy nodes (beige), it encounters. As every forwarding node includes a number of proxy nodes, a forwarding node defines a “corridor” for the returning message. Thus, the network is not simply flooded with messages containing the answer to a query. Instead, the route to the final *Responder* already defines a good route back to the initial *Client*.

IV. Conclusion and Future Work

Mergenet is a system that assists user to socialize in way that virtual social network cannot do. It helps them to make contacts and lets them use these contacts to share and

Algorithm 3 Query & Response Handling

```
1: if $c.isExpert(q.getTopic())$ then
2:   $q.setAnswer()$
3:   $q.setIsQuery($false$)$
4: end if
5: $c.Q.append(q)$
```

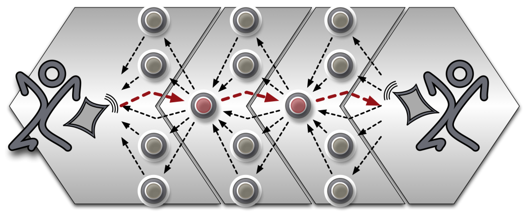


Fig. 4: Illustration of the message protocol.

gather information. To the best of our knowledge, our design approach is the first to enable users in MANETs or DTNs to get a reply for a submitted query without flooding the network with messages. Furthermore, our message protocol introduces some simple privacy preserving mechanics due to the message design, however, there is much more to be done to call *Mergenet* very secure. Actually we are implementing this system, running first tests on how to identify *virtual friends* as well as optimizing the way we aggregate the Bloom filters of several users without running into the problem of achieving a high false positive rate. This also involves compression of Bloom filters without loosing their efficiency.

References

[1] “Mit media lab: Reality mining,” `http://reality.media.mit.edu`.

[2] D. Bauer, P. Hurley, R. Pletka, and M. Waldvogel, “Bringing efficient advanced queries to distributed hash tables,” *lcn*, vol. 00, pp. 6–14, 2004.

[3] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[4] E. M. Daly and M. Haahr, “Social network analysis for routing in disconnected delay-tolerant manets,” in *MobiHoc ’07: Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*. New York, NY, USA: ACM, 2007, pp. 32–40.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2007 – 36th printing.

[6] M. Girvan and M. E. Newman, “Community structure in social and biological networks.” *Proc Natl Acad Sci U S A*, vol. 99, no. 12, pp. 7821–7826, June 2002. [Online]. Available: http://dx.doi.org/10.1073/pnas.122653799

[7] Y. Gong, Y. Xiong, Q. Zhang, Z. Zhang, W. Wang, and Z. Xu, “Anycast routing in delay tolerant networks,” in *GLOBECOM’06: In Proceedings of the Global Telecommunication Conference 2006*, San Francisco, CA, USA, November 2006, pp. 1 – 5.

[8] D. Guo, J. Wu, H. Chen, and X. Luo, “Theory and network applications of dynamic bloom filters,” in *INFOCOM*, 2006.

[9] P. Hurley and M. Waldvogel, “Bloom filters: One size fits all?” *lcn*, vol. 0, pp. 183–190, 2007.

[10] A. G. Miklas, K. K. Gollu, K. K. Chan, S. Saroiu, K. P. Gummadi, and E. de Lara, “Exploiting social interactions in mobile systems,” Innsbruck, Austria, September 2007.

[11] S. Milgram, “The small world problem,” *Psychology Today*, vol. 1, pp. 60 – 67, May 1967.

[12] M. Mitzenmacher, “Compressed bloom filters,” in *PODC ’01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2001, pp. 144–150.

[13] D. P. Reed, “That sneaky exponential—Beyond Metcalfe’s law to the power of community building,” `http://www.reed.com/Papers/GFN/reedslaw.html`, 1999.

[14] S. Rhea and J. Kubiatowicz, “Probabilistic location and routing,” in *INFOCOM 2002. Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2002, pp. 1248 – 1257.

[15] W. Zhao, M. Ammar, and E. Zegura, “Multicasting in delay tolerant networks: semantic models and routing algorithms,” in *WDTN ’05: Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. New York, NY, USA: ACM, 2005, pp. 268–275.