Scalable Opportunistic Forwarding Algorithms in Delay Tolerant Networks using Similarity Hashing

Cong Liu[†], Yan Pan[†], Ai Chen[‡], Kaigui Bian[♯], and Jie Wu[◇]
[†]Sun Yat-sen University, [§]Shenzhen Institutes of Advanced Technology CAS,
[♯]Peking University, [◇]Temple University
Email: liucong3@mail.sysu.edu.cn, panyan5@mail.sysu.edu.cn, ai.chen@siat.ac.cn,
bkg@pku.edu.cn, jiewu@temple.edu

***Abstract*—Due to intermittent connectivity and uncertain node mobility, opportunistic message forwarding algorithms have been widely adopted in delay tolerant networks (DTNs). While existing work proposes practical forwarding algorithms in terms of increasing the delivery rate and decreasing data overhead, little attention has been drawn to the control overhead induced by the algorithms. The control overhead could, however, make the forwarding algorithms infeasible when the network size scales. In this paper, we are interested in increasing scalability by reducing control overhead, while retaining the state-of-the-art forwarding performances. The basic idea is to use locality-sensitive hashing to map each node as a hash-code, and use these hash-codes to compute the pair-wise similarity that guides the forwarding decisions. The proposed SOFA algorithms have reduced control overhead and competitive forwarding performance, which are verified by extensive real trace-driven simulations.**[1]

***Index Terms*—Delay Tolerant Networks, Opportunistic Forwarding, Scalability, Similarity, Simulation.**

I. Introduction

Delay tolerant networks (DTNs) [1] are sparse mobile networks, where connectivity amongst the nodes is intermittent. A contemporarily connected source-destination path may not exist between any pair of source and destination nodes at any time. Designing forwarding (routing) algorithms in DTNs is challenging, since traditional connection-based routing is not applicable, and messages need to be delivered in a store-carry-forward paradigm. Examples of DTNs include mobile social software [2], pocket switch networks [3], vehicle and pedestrian networks [4], low duty cycle sensor networks [5], deep space satellite networks, underwater acoustic buoy networks, and many dedicated networks for developing regions.

Due to uncertainty in node mobility, a multi-copy opportunistic scheme is usually adopted: each DTN node opportunistically spawns copies, while identical copies are simultaneously kept by multiple nodes [4], [6]. The message is delivered if one of these nodes encounters the destination. In these algorithms, the two contradictory objectives are:

1) High forwarding performance. Forwarding algorithms seek to improve the percentage of generated messages being sent to their destinations within their time-to-lives. Different *forwarding metrics* are proposed, e.g., the frequency that each node encounters the destination, to select forwarding nodes.

2) Low overhead. DTN nodes are typically power-constrained devices. Power is consumed when a forwarding algorithm computes forwarding decisions, and stores and forwards messages. Algorithms that make thrifty forwarding decisions and that require low computation and storage complexity are desired.

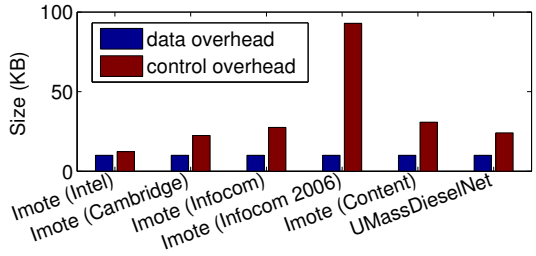


Fig. 1. Data overhead versus control overhead (in terms of the contact frequency information exchanged among the nodes). Assuming that each message is 1KB, and 10 copies are spawned for each message. A 4 bytes contact frequency regarding each destination needed to be obtained from each encountered node. The time-to-live of the messages span the entire trace. The average per node contacts in the above traces are around 307, 561, 688, 2323, 770, and 603, respectively. Refer to Section VI for details about the trace.

An enormous amount of research work has been devoted to opportunistic forwarding algorithms, e.g., [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], which reduces data overhead, in terms the average number of copies (or forwarding times) per message, while retaining a high routing performance, in terms of delivery rate or delay. However, methods used to combat the control overhead are rarely discussed in the literature, due to the intuition that data overhead always dominates.

As illustrated by the comparison between the two types of overhead in Figure 1, the overhead of control messages is significant and should not be overlooked. For example, assume that 10 copies of a 1KB data message are spawned to 10 forwarding nodes. Each of these nodes encounters 1,000 other nodes during the time-to-live of the message. The forwarding algorithm is required to obtain a 4 bytes contact frequency from each encountered node. Then, the per message data overhead is only 10KB in total, but the per message control overhead, in terms of the contact frequencies exchanged alone, is 40KB!

In this paper, we take the first step in making DTN opportunistic forwarding algorithms more scalable through

[1]This work was funded in part by National Science Foundation of China (grant No. 61370021, 61003045, 61003296, 61201245, 61003241), Natural Science Foundation of Guangdong Province, China (grant No. S2013010011905), Shenzhen Overseas High-level Talents Innovation and Entrepreneurship Funds under Grant No. KQC201109050097A, NSF grants ECCS 1231461, ECCS 1128209, CNS 1138963, CNS 1065444, and CCF 1028167.

reducing the control overhead, while retaining competitive forwarding performances. Specifically, we are interested in reducing the control overhead by compressing the amount of routing information exchanged among the nodes. We make the following efforts to realize this goal:

1) We propose using *locality-sensitive hashing* (LSH) to generate a hash-code for each node, such that the pair-wise similarities can be computed among these hash-codes.

2) We investigate different ways to generate LSH hash-codes, assuming different levels of knowledge about the network, from global to strictly local.

3) We address several practical implementation issues related to the forwarding algorithms that use LSH hash-codes.

4) We conduct an extensive simulation study with multiple contact traces to evaluate the feasibility and necessity of the proposed SOFA forwarding algorithms.

We will first review some preliminaries on DTN opportunistic forwarding algorithms and LSH algorithms in Section II. Then, in Sections III and IV, the proposed *scalable opportunistic forwarding algorithm* (SOFA) will be presented, and different LSH algorithms that preserve pair-wise similarity as well as several related implementation issues will be discussed. Lastly, extensive simulations and their results will be presented and analyzed in Section VI.

II. Preliminaries

A. DTN opportunistic forwarding algorithms

Opportunistic forwarding is usually applied in power constrained DTNs with nondeterministic node mobility. Flooding-based algorithms [17] have large data forwarding overheads. Opportunistic forwarding algorithms try to forward messages to a small set of nodes with high delivery potential. In multi-copy forwarding schemes, identical copies of the same message are simultaneously kept by multiple nodes to combat the uncertain mobility.

As an example, in the *delegation forwarding* [18] algorithm, message copies are only allowed to be forwarded from a node of a lower forwarding quality to a node with a higher quality, as to avoid data overhead. To further reduce data overhead, once a forwarding node has forwarded a copy to another node of a higher forwarding quality, say q, it will increase its threshold for the future forwardings to q. The general concept of forwarding quality can be instantiated as any forwarding metric, such as contact frequency [7].

B. Similarity metrics

Forwarding metrics can be generalized as similarity metrics, e.g., the similarity between a node and the destination. Similarities can be divided into two categories: pair-wise similarity and social similarity. A pair-wise similarity describes the relationship between a pair of nodes, such as encounter frequency [7], time elapsed since last encounter [4], [19], and boolean pair-wise similarity [11]. A social similarity describes one or more well-defined social features of a node, such as social betweenness [11], community label [12], centrality rank [12], geometric similarity [13], location similarity [14], and vectorized features [20].

For message forwarding algorithms, pair-wise similarity metrics are more direct and precise, but they are more expensive to maintain. For instance, an encounter frequency-based forwarding algorithm may need to store and exchange a full vector of encounter frequencies of the current node and the other nodes proactively or reactively. On the other hand, social similarity metrics usually rely on centralized computation and distribution, which might not always available. Moreover, they relay on the assumption of *correlated interaction* [12]. In sociology, the idea of correlated interaction is that a node of a given community is more likely to interact with another node within the same community than with a randomly chosen node.

Since DTNs and their applications display a wide range of characteristics, there is no single best similarity metric for all DTNs or application scenarios. Our similarity metrics, i.e., the hash-codes, generated by the proposed LSH algorithms essentially belong to social similarity metrics, and inherit their characteristics. However, the proposed LSH algorithms can use any vectorized pair-wise similarity and social similarity metric as their inputs, and a centralized computation is optional. For instance, we will use a node's vector of encounter frequencies with all other nodes as the per-node input to our algorithm. Encounter frequencies is widely used in literature and they can be easily summarized in various trace data.

C. Locality-sensitive hashing (LSH)

LSH is a class of dimensionality reduction techniques, which maps high-dimensional vectors to small-sized hash-codes. A property of LSH is that similar input vectors have similar or identical hash-codes, which is atypical of hash-functions. Consider two vectors that differ in a single element. Cryptographic hash-functions, such as SHA-1 and MD5, will hash these two vectors into two completely different hash-values. However, LSH will hash them into similar or identical hash-values. LSH has been widely applied in nearest neighbor search, near-duplicate detection, hierarchical clustering, etc.

As an example, the *random projection* [21] method of LSH chooses k random hyperplanes r_i ($1 \leq i \leq k$) at the outset, and uses them to compute the hash-code of each input vector v. The i-th bit in the hash-code of v depends on the sign of $v \cdot r_i$, which in turn depends on which side of the hyperplane r_i vector v lies. For two vectors v_1 and v_2, the probability that they lie on the same side of a hyperplane r_i equals $1-\frac{\theta(v_1,v_2)}{\pi}$, where $\theta(v_1, v_2)$ is the angle between v_1 and v_2, and $1-\frac{\theta(v_1,v_2)}{\pi}$ is closely related to $cos(\theta(v_1, v_2))$. When k is large enough, the percentage of common 1s in the hash-codes of v_1 and v_2 provides a good estimation of $cos(\theta(v_1, v_2))$, which can be used to measure the similarity between v_1 and v_2.

III. Scalable Opportunistic Forwarding Algorithm (SOFA)

A. Assumptions

SOFA is proposed for large and power constrained DTNs. SOFA uses the hash-codes of the nodes to compute their pair-wise similarities. The input of the proposed LSH algorithms is a vector of *features* per node, which can be any vector of social features, or any vectorized pair-wise similarities. For ease of illustration and simulation, in this paper, we use the vector of encounter frequencies.

Algorithm 1 SOFA

```
c ← the current node
e ← a node that c encounters
h_c ← the hash-code of c
h_e ← the hash-code of e requested from e
for each message m carried by c
    h_m ← the hash-code of the destination of m
    τ ← the forwarding threshold of m
    if τ is not defined
        τ ← s(h_c, h_m)      // s is a similarity function
    if τ < s(h_e, h_m)
        τ ← s(h_e, h_m)
        send a copy of m to e
```

Specifically, we assume that node IDs can be mapped to consecutive indexes starting from 1, so that we can maintain a vector per node of its contact frequencies with other nodes, ordered by node indexes. In practice, such a mapping can be implemented by hashing node IDs to integer indexes, with conflicts solved arbitrarily.

Secondly, since SOFA needs the LSH hash-codes of a forwarding node and a destination to derive their similarity, we assume that each message is stamped with the hash-code and the ID of its destination. Similar assumptions about the hierarchical addresses, the locations, and the community labels are widely made in hierarchical routing, location-based routing, and community-based routing, respectively.

For a routing problem, we always assume that the source knows some information, such as node ID, about the destination a priori. In the case that the hash-codes are not known a priori or are constantly updated, we propose a *destination hash-code service* in Section IV-A that allows the source to look up the hash-codes of the destinations. This approach is similar to the *location service* in location-based routing.

B. Problem statement

Our main task is to calculate an LSH hash-code for every node, such that the pair-wise similarity between each pair of nodes can be computed from their hash-codes.

Let Y be an $n \times m$ matrix, where n is the number of nodes and m is the number of features. The i-th row of Y, denoted by Y_i, is the vector of features of the i-th node in the network. One can measure the similarity between two feature vectors, Y_i and Y_j, using their inner product, $\langle Y_i, Y_j \rangle = \sum_u Y_i^{(u)} \times Y_j^{(u)}$, where $Y_i^{(u)}$ is the u-th element in vector Y_i. Geometrically, the inner product of two vectors is the product of the magnitudes of the two vectors and the cosine of the angle between them. The similarity $\langle Y_i, Y_j \rangle$ is similar in spirit to the social similarity in [11], which is calculated using a binary vectors, and the one in [12], where two nodes are similar if they belong to same k-clique communities.

Our objective is to compute n hash-codes $x_1, x_2, \cdots, x_n$, such that, for any tuple of indexes i, j, and k, if $\langle Y_i, Y_k \rangle > \langle Y_j, Y_k \rangle$, then $s(x_i, x_k) > s(x_j, x_k)$ for some similarity function s with high probability. In this paper, we will solve the above problem in the three different scenarios, respectively:

1) There is a *manager*, e.g., one of the nodes, that knows the entire matrix Y. The manager computes all $x_1, x_2, \cdots, x_n$, and broadcasts them to every node.

2) No centralized mechanism exists. Each node computes x_i independently with its local information Y_i.

3) There is a *manager* that knows some, but not all, rows of the matrix Y, and is able to broadcast a small amount of information in the network. Each node i then computes x_i with this broadcast information and its local information Y_i.

We will provide the solutions in these three scenarios in Sections III-E, III-F, and III-G, respectively.

C. Forwarding algorithm

Let us first explain the framework of the SOFA algorithm, provided that every node has obtained its hash-code for the time being. SOFA is based on *delegation forwarding* [18], which was briefly introduced in Section II-A. SOFA is listed in Algorithm 1, and is explained in the following.

Algorithm 1 will be executed on every node c, whenever it encounters another node e. The hash-code h_e of e can be either piggybacked to the hello messages of e, retrieved from the local repository of c, or requested from e using a control message. Note that, except for the last case, the only control message required by SOFA is the inevitable hello messages. After that, c looks over every message m, which it is carrying, for possible forwarding to e.

A forwarding threshold τ is maintained for each message m. If the similarity $s(h_e, h_m)$ between h_e and h_m (h_m is the hash-code of the destination of m), is larger than τ, then m will be forwarded to e while τ is increased to $s(h_e, h_m)$.

D. Ideal similarity

An ideal and trivial way to represent each node i is to use the full vector Y_i. With this "lossless" representation, we can simply use $\langle Y_i, Y_k \rangle$ as the similarity function, and a benchmark to evaluate the other LSH-based similarities to be introduced. Note that this representation of nodes is impractical, since each encountering peer needs to send and receive a full vector of size $O(n)$, instead of a short hash-code.

E. Centralized LSH with low-rank matrix decomposition

In the first scenario, there is a *manager* who knows the entire matrix Y and computes all $x_1, x_2, \cdots, x_n$.

1) Simplified precise decomposition: We temporarily assume that the rank of Y is a small number k. That is, Y can be decomposed as the multiplication of two other matrixes:

$$Y = X\Theta, \tag{1}$$

where X is an $n \times k$ matrix, Θ is a $k \times m$ matrix. This decomposition is illustrated in Figure 2.

Denote Y_i as the i-th row of Y, and $Y_i^{(j)}$ as the j-th element on row Y_i. Equation 1 is equivalent to the following two equations.

$$Y_i = X_i\Theta,\ 1 \le i \le n,$$

$$Y_i^{(j)} = \langle X_i, \Theta_j^T \rangle,\ 1 \le i \le n,\ 1 \le j \le m,$$

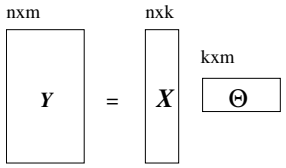


Fig. 2. Low-rank decomposition of $Y = X\Theta$. n is the number of nodes, m is the number of features representing each node, and k is the number of bases in Θ, which generates k small integers that are combined to form the LSH hash-codes.

where Θ^T is the transpose of Θ.

Theorem 1 below shows that if we use X_i as the hash-code of node i, the objective described in Section III-B is satisfied. The orthonormality of the rows of Θ is defined as: (1) each row Θ_i is normalized, i.e., $\langle \Theta_i, \Theta_i \rangle = 1$, and (2) each pair of rows are orthogonal, i.e., $\langle \Theta_i, \Theta_j \rangle = 0$ if $i \neq j$. In other words, $\Theta\Theta^T = I$, where I is the identity matrix.

Theorem 1: If $Y = X\Theta$ and the rows of Θ are orthonormal, $\langle Y_i, Y_k \rangle > \langle Y_j, Y_k \rangle \Leftrightarrow \langle X_i, X_k \rangle > \langle X_j, X_k \rangle$

Proof: It is sufficient to prove that $\langle Y_i, Y_k \rangle = \langle X_i, X_k \rangle$.

$$\begin{aligned} \langle Y_i, Y_k \rangle &= X_i\Theta(X_k\Theta)^T = X_i\Theta\Theta^T X_k^T \\ &= X_i(\Theta\Theta^T)X_k^T = X_i I X_k^T = \langle X_i, X_k \rangle. \end{aligned}$$

■

If the rows of Θ are not orthonormal vectors, we can construct a new matrix Θ'' from Θ such that the rows of Θ'' are orthonormal. There are algorithms that make orthonormal bases, such as the QR algorithm and the Gram-Schmidt Algorithm. Since each row in Θ is a linear combination of the rows in Θ'', for each $Y_i = X_i\Theta$, there exists an X_i'', such that $Y_i = X_i''\Theta''$.

2) Low-rank and compact hash-code approximation: We used X_i as the hash-code for node i. However, X_i contains k real numbers, which is not a compact hash-code. To solve this problem, we make two changes to Equation 1.

a) Instead of using a vector X_i of k real numbers to represent node i, we replace X_i by $\alpha^{(i)}Z_i$, where $\alpha^{(i)}$ is a single real number and Z_i is a vector of k small integers. In this paper, $Z_i^{(j)} \in \{-1, 0, 1, 2\}$ for $1 \leq j \leq k$. With this small range, it requires two bits to represent each $Z_i^{(j)}$ and $2k$ bits are to represent Z_i.

b) Y may not be low-rank in reality. Now, we define the number of bases k, as illustrated in Figure 2, as a parameter in our algorithm, which is not the large rank of Y anymore. For a k much smaller than the rank of Y, we choose an $n \times 1$ column vector α, an $n \times k$ matrix X and a $k \times m$ matrix Θ such that $\alpha : X\Theta$ approximates Y, where operator : is the element-wise multiplication operator. The i-th rows in the result of $\alpha : X$ is $\alpha^{(i)}X_i, 1 \leq i \leq n$.

To measure the difference between Y and $\alpha : X\Theta$, we define

$$\textit{diff} = \frac{||Y - \alpha : X\Theta||_F^2}{||Y||_F^2}, \tag{2}$$

where, the Frobenius norm $||\cdot||_F^2$, or the Euclidean norm of Y is defined as $||Y||_F^2 = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}|Y_i^{(j)}|^2}$.

To minimize *diff*, we have the following optimization problem, which solves for the α and X we need, with Θ being a byproduct.

$$\begin{aligned} \min_{a,Z,\Theta} \quad & ||Y - \alpha : Z\Theta||_F^2, \\ s.t. \quad & Z_i^{(j)} \in \{-1, 0, 1, 2\}, 1 \leq i \leq n, 1 \leq j \leq k, \\ & \Theta\Theta^T = I. \end{aligned} \tag{3}$$

This problem is a *mixed integer and linear programming* (MILP), which can be solved by the *branch and cut* algorithm and other greedy heuristics. We simply solve this problem by first solving its LP relaxation with a bounded X and then applying a greedy local search.

Finally, the manager node uses $\alpha^{(i)}Z_i$ as the hash-code of node i and broadcasts them to every node in the network. Each node i can use $\langle \alpha^{(i)}Z_i, \alpha^{(d)}Z_d \rangle$ to compute its similarity to any destination d.

F. Localized LSH with random bases

In the second scenario, no centralized mechanism exists, and each node computes its hash-code independently with its local information Y_i.

Note that the optimization problem shown in Equation 3 chooses $\alpha^{(i)}$ and Z_i $(1 \leq i \leq n)$ such that $\alpha^{(i)}Z_i\Theta$ approximates Y_i. At the same time, it chooses Θ such that all Y_i $(1 \leq i \leq n)$ can be approximated with a minimum difference defined in Equation 2. Therefore, the rows in Θ is a set of principal components of the rows in Y, which means that any Y_i can be approximated by a linear combination of the rows in Θ.

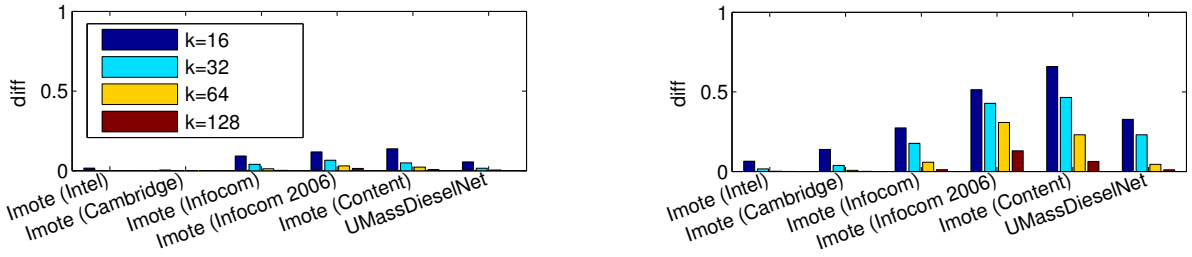
However, such principal component bases Θ cannot be computed without a global knowledge of Y. In this second scenario, we use randomly generated bases Θ instead. Similarly, we make the rows in Θ orthonormal, in order that our objective in Section III-B is guaranteed with high probability.

With these random bases in Θ being predetermined, the hash-code $\alpha^{(i)}Z_i$ for each node i can be computed with its local information Y_i by

$$\begin{aligned} \min_{a^{(i)},Z_i} \quad & ||Y_i - \alpha^{(i)}Z_i\Theta||_F^2, \\ s.t. \quad & Z_i^{(j)} \in \{-1, 0, 1, 2\}, 1 \leq j \leq k. \end{aligned} \tag{4}$$

Note that, if we force that $Z_i^{(j)} \in \{0, 1\}, 1 \leq j \leq k$, this method is similar to the *random projection method*, which we discussed in Section II-C.

Figure 3 compares the two hash-code methods in Sections III-E and III-F, respectively, in terms of *diff*s. The *diff* between Y and the reconstruction of $\bar{Y} = \alpha : X\Theta$ is defined in Equation 2. We generate a matrix Y from each of the six contact traces that we will use in our simulation, where Y_i is the vector of contact frequencies for node i. The results show that, in terms of *diff*, the centralized hash-codes using learned bases are much better than those computed using random bases. Fortunately, similar to the observations in



(a) Centralized LSH with low-rank matrix decomposition

(b) Localized LSH with random bases

Fig. 3. Reconstruction difference *diff* of Y using α and X with different numbers of bases k. *diff* is defined in Equation 2 .

random projection, as the number of bases increases, the *diff*s decrease significantly in all traces.

The approximation error may invalidate the nice property that $\langle Y_i, Y_k\rangle > \langle Y_j, Y_k\rangle \Leftrightarrow \langle \bar{Y}_i, \bar{Y}_k\rangle > \langle \bar{Y}_j, \bar{Y}_k\rangle$, which results in forwarding to nodes of low delivery potential. The actual effect is hard to analyze theoretically because it depends on the distribution of the contact frequencies and the correlation among the node mobility. Simulations will be performed in Section VI to compare the performance of the forwarding algorithms that use different hash-code.

G. Semi-localized LSH with learned bases

The third scenario is a more realistic one, which requires a *manager* to collect some, but not all, information about the nodes in the network, i.e., only some of the rows in the entire matrix Y are available for the manager. Also, the manager is allowed to broadcast some information in the network. The actual amount of information that the manager collects and broadcasts is a parameter in practical deployment.

The comparison in Figure 3 shows that using a learned bases can generate better hash-codes than using random bases. Our approach is to let the manager calculate a set of bases that captures the principal components of the partially collected feature vectors in Y. There are two steps in this approach.

1) The manager uses the collected feature vectors as rows to forms a shorter matrix Y. Secondly, it uses the optimization function in Equation 3 to solve for a set of bases Θ. Then, it broadcasts Θ.

2) Each node uses the Θ learned by the manager, its local feature vector Y_i, and the optimization function in Equation 4 to solve for its $\alpha^{(i)}$ and Z_i.

In the case that the number of learned bases allowed to broadcast due to bandwidth limitations is not large enough to generate hash-codes of desired quality, random bases can be added to the learned bases to form a larger set of bases.

It can be proven that the algorithms to generate hash-codes in this section belong to the LSH family using similar technique with which random projection method is proven.

IV. Implementation Issues

A. Destination hash-code service

Assume that a source node has the ID of a destination, the *destination hash-code service* to be presented can help to obtain the hash-code of the destination. The idea is similar to the *location service* used in *location-based routing*.

In fact, SOFA does not forward messages towards IDs, but towards hash-codes. With this fact, the *service* uses a globally known hash function H that hashes node IDs to node hash-codes. Note that H is a regular hash function instead of an LSH function, which means that similar IDs will be hashed to very different hash-codes. The *service* is a distributed and cooperative algorithm, which consists of two parts:

Registration: After computing its hash-code h_i, each node i, sends the pair $\{i, h_i\}$ to the node whose LSH hash-code is the most similar to $H(i)$ (Assume that there is a similarity function to compare the two types of hash-code). Note that there might not exist a node whose hash-code is exactly $H(i)$. After receiving a pair $\{i, h_i\}$, each node j stores the pair, until it finds another node k such that h_k is more similar to $H(i)$ than h_j, and sends the pair to node k .

Query: A node j that needs to know h_i will send the query $\{i\}$ together with j's hash-code h_j to the node whose hash-code is the most similar to $H(i)$. Any node that has the pair $\{i, h_i\}$ will send it to j via h_j.

B. Asymmetric similarities

We have assumed that each node i can be represented by a vector of features Y_i, and the similarity $s_{i,j}$ between nodes i and j is $\langle Y_i, Y_j\rangle$. Under this definition, similarities are symmetric: $s_{i,j} = s_{j,i}$. However, this is not always true: for instance, contacts are directional by definition and they seldom occur pair-wisely in real traces.

To represent asymmetric similarities, we represent each node i with two vectors of features Y_i^s and Y_i^r. With this *two vectors per node* representation, the asymmetric similarities between nodes i and j are now $s_{i,j} = \langle Y_i^s, Y_j^r\rangle$ and $s_{j,i} = \langle Y_j^s, Y_i^r\rangle$. For the contact frequency features used in this paper, Y_i^s is the *sending contact frequencies* of node i, which is summarized over the past contacts where i is able to send messages. The vector for the *receiving contact frequencies* Y_i^r is computed likewise.

C. Feature enhancing

For specific type of features, we may perform practical enhancement such that they result in more accurate similarities. Specifically, we enhance the each vector of contact frequencies by setting $Y_i^{r(i)} = \sum_{j\neq i} Y_i^{r(j)}$. This is a reasonable enhancement because $Y_i^{r(i)}$, i.e., the sending frequency of i to itself, is

originally unmeasurable, and the fact that node i can always send messages to itself more frequently than the sum of all other nodes. Y_i^r is enhanced likewise.

D. Message vectors

In a typical opportunistic forwarding algorithm, each encountering peer i sends and receives the following control messages: (1) a message *summary vector* containing the IDs of the messages in i's buffer, which prevents the other peers from sending the messages that i has; (2) a query and a reply for the peer's similarity with the destination of each message in i's buffer; and (3) other optional messages, such as the disseminated acknowledgments for the successfully delivered messages [4]. We focus on removing the summary vector in this subsection.

Summary vector is used in almost all DTN opportunistic forwarding algorithms. However, summary vectors may produce an extraordinarily high volume of control overhead, which will be shown in Section VI. In SOFA, we optionally remove the use of summary vector. The disadvantage of removing summary vectors is that message copies will be forwarded redundantly. Fortunately, SOFA is based on *delegation forwarding*, which requires that each message must be forwarded to nodes with increasing similarities to the destination. This prevents most redundant forwarding and loops. An analysis of the data overhead after the removal of summary vector will be presented in the following.

V. Analysis

A. Data forwarding overhead

We will derive an upper bound for the expected number of forwardings per message in Algorithm 1 with and without the use of *summary vector*, respectively. The derivation is similar in spirit to [18]. We imagine that there are two stages in the forwarding of each message. In the first stage, copies of the message are forwarded only to the k nodes that do not have the message. In the second stage, copies of the message are forwarded redundantly among these k nodes. Note that (1) each forwarding of a copy will increase its threshold, and both stages 1 and 2 will eventually stop after the thresholds in all copies reach the maximum similarity value, and (2) stage 2 overlaps stage 1 in reality, which may decrease k. As a result, our analytical two stages will have a larger number of forwardings than the overlapping stages in reality.

In stage 1, we assume that the forwardings happen in s steps (sub-stages). In each step i ($1 \leq i \leq s$), there is a corresponding forwarding set C_i. Initially, C_1 only contains the source. In each step i, each node in C_i is expected to forward a copy to a new node, and $E[|C_{i+1}|] = 2E[|C_i|]$ due to the assumption that no redundant forwarding exists in stage 1. Let τ_i be the expectation of the similarity threshold on the message copies in C_i, and let expectation $g_i = 1 - \tau_i$ be the gap between τ_i and the maximum similarity. Assuming that the similarities are uniformly distributed between [0,1], then $g_{i+1} = \frac{g_i}{2}$ and $g_0 = \frac{1}{2}$, which follows $g_i = \frac{1}{2^i}$. Note that, in [18], it is argued that the number of forwardings is determined by number of distingue similarities, rather than by the distribution of the similarities. Let N be the set of nodes, and $n = |N|$. Define $C_i' \subseteq (N - C_i)$ as the set of nodes with similarities higher than g_i, then $E[|C_i'|] = g_i n = \frac{n}{2^i}$ and the new forwarding nodes created by step i must in C_i', i.e., $(C_{i+1} - C_i) \subseteq C_i'$. This follows that stage 1 is expected to stop when $|C_i'| \leq |C_i|$. Since $E[|C_s'|] = \frac{n}{2^s}$ and $E[|C_s|] = 2^s$, we have $s \leq \frac{\log_2 n}{2}$, and the number of forwardings in stage 1 is:

$$k = |C_i| = 2^s < 2^{\frac{\log_2 n}{2}} = \sqrt{n}.$$

In stage 2, we assume that the forwardings happen in s' steps (sub-stages) among the $k = \sqrt{n}$ nodes. In each step i ($1 \leq i \leq s'$), except the node with the maximum similarity, all nodes forward and obtain higher similarity thresholds for the message. Again, let τ_i be the expectation of the similarity thresholds of the message copies in step i, and let expectation $g_i = 1 - \tau_i$ be the gap between τ_i and the maximum similarity. Similar to stage 1, we have $g_{i+1} = \frac{g_i}{2}$, and $g_1 = \frac{1}{2}$, and therefore $g_{s'} = \frac{1}{2^{s'}}$. Since $k g_{s'} = 1$ (stage 2 stops when all similarities are equal), we have $s' = \log_2 k$, and the number of forwardings k' in the two stages is, in total:

$$k' < k s' = k \log_2 k = \sqrt{n} log_2 \sqrt{n}.$$

The analysis above assumes that all nodes have different similarities with the destination, and all of the n nodes can encounter each other, thus giving an upper bound on the expected number of forwardings.

To summarize, the average number of forwardings per message in SOFA is bounded by $\sqrt{n}$ when summary vectors are used, and it is bounded by $\sqrt{n} \log_2 \sqrt{n}$ without using summary vectors.

B. Computation overhead

All the centralized, localized, and semi-localized calculations of hash-code, as presented in Sections III-E to III-G, can be conducted offline. SOFA only need to compute the similarity between the nodes using their hash-codes, which is simple to calculate the inner-product of each pair of hash-codes.

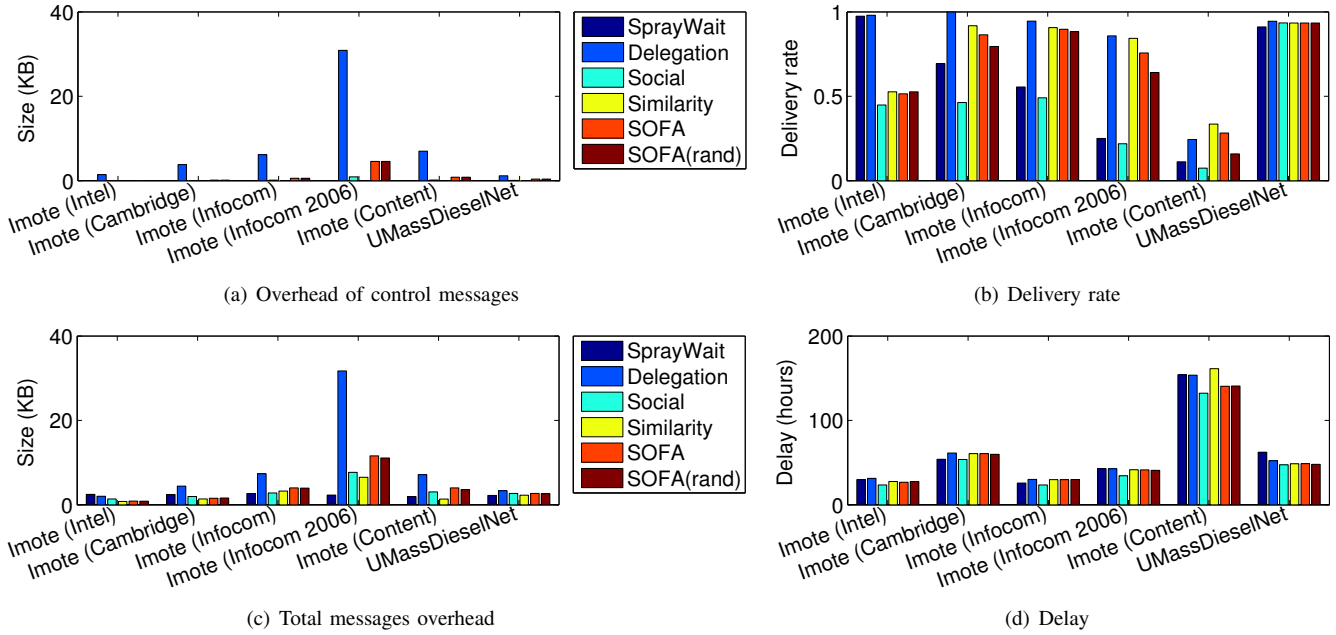
VI. Simulation

A. Simulation trace and method

We conduct trace-driven simulations on six real contact traces. They include the UMassDieselNet trace [4], which consists of 55 days of the bus-to-bus contacts (holidays are removed due to reduced schedules), and the Cambridge Haggle trace set [22], which includes a total of five traces of Bluetooth device contacts by people carrying mobile devices (iMotes). Bluetooth contacts are classified into internal contacts between internal iMotes, and external contacts between internal iMotes and the external bluetooth devices overheard. Since there is no record of contacts between external nodes, we only use

TABLE I. Information about the trace data.

Trace	Contact	Length (d,h:m.s)	Fwd nodes	Dest nodes
Imote (Intel)	2766	4,3:46.31	9	128
Imote (Cambridge)	6732	6,1:6.28	12	223
Imote (Infocom)	28216	2,22:52.49	41	264
Imote (Infocom 2006)	227657	3,21:43.39	98	4519
Imote (Content)	41587	23,19:50.18	54	11421
UMassDieselNet	18700	28,17:12.51	31	32



(a) Overhead of control messages

(b) Delivery rate

(c) Total messages overhead

(d) Delay

Fig. 4. Simulation results on all six traces in terms of control overhead, total overhead, delivery rate and delay. *Similarity* is an ideal forwarding algorithm.

internal nodes as source nodes and forwarding nodes, but allow all nodes to be destinations. Information about the traces are summarized in Table I.

In each simulation, we generate 5,000 messages. Each message is assigned a random internal node as the source, and another random destination among all nodes. All messages are created on their sources at time 0. Results are measured at 10 time-to-live points of 10%, 20%, $\cdots$, 100% of the total trace time, respectively. Although infinite buffer size and bandwidth might cause inaccuracies [23], our simulations are conducted with unrestricted buffer sizes per node, and unlimited number of messages sent per contact, since otherwise some buffer manager policy and queuing policy would need to step in, which could make the results more difficult to analyze. The forwarding algorithms to be compared are: *SprayWait* [6]. It blindly forwards a number of L copies per message. In our simulations we set $L = 10$. *Delegation* [18]. As introduced in Section II, it uses contact frequency as its forwarding quality metric. *Social*. It measures social centrality of each node in terms of its total contact frequency, and uses the delegation forwarding framework. *Similarity*. It uses the ideal bench-mark similarity in Section III-D. *SOFA*. As presented in Section III-E, it uses the hash-codes generated with at most 64 learned bases. Sometime we use less than 64 bases because the number of orthonormal bases cannot exceed the number of features. *SOFA(rand)*. As presented in Section III-F, it uses the hash-codes generated with at most 64 random bases.

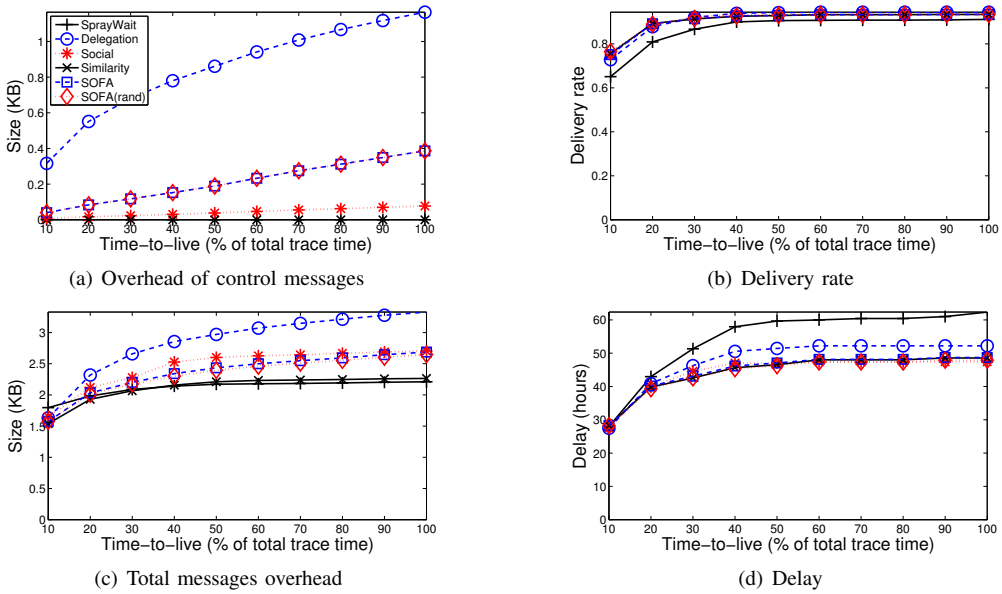
We leave the implementation of the algorithm using semi-localized hash-code, presented in Section III-G, to our future work. Its performance should stands somewhere between those of *SOFA* and *SOFA(rand)*. For simplicity, we assume that the source of each message knows the hash-code of the corresponding destination, without needing a destination hash-code service.

B. Results and analysis

We first conduct simulations without using summary vectors. Figure 4 shows the simulation results at the end of each of the six simulations. To show the simulation results at different times during simulation, we show the simulation results on the UMassDieselNet trace, with different message time-to-live values, in Figure 5.

1) Overhead of control messages: It is the sum of the size of the two types of control messages: routing information and summary vectors. In this subsection, summary vectors are not used. Routing information is different for different forwarding algorithms. *SprayWait* does not use any routing information. Assuming each ID or contact frequency is represented by 4 bytes, each node in *Delegation* requires to send and receive $8d$ bytes of routing information, where d is the number of different destinations for all the messages in the node. *Social* sends a single total contact frequency of 4 bytes per contact. We assume that the ideal *Similarity* does not need to send any routing information, which otherwise will be in large volume. Finally, *SOFA* and *SOFA(rand)* use a fixed hash-code of 20 bytes per node, 4 bytes for α and 2 bits for each of the 64 Z_is. As shown in Figure 4(a), the control overhead of *SOFA* and *SOFA(rand)* are much smaller than *Delegation*.

2) Delivery rate: Delivery rate measures the percentage of messages that research their destinations before a given time-to-live. As shown in Figure 4(b), in traces Imote (Infocom) and UMassDieselNet, the delivery rates of *SOFA* and *SOFA(rand)* are very close to Delegation, the best one. In the two largest traces Imote (Infocom 2006) and Imote (Content), the delivery rates of *SOFA* and *SOFA(rand)* are slightly worse than Delegation, while significantly better than the others. This is because of the relatively large *diff*s in these traces, as shown in Figure 3. Considering the fact that the control overheads of *SOFA* and *SOFA(rand)* are just a tiny fraction of *Delegation*, *SOFA* and *SOFA(rand)* provide very good trade-offs between



(a) Overhead of control messages

(b) Delivery rate

(c) Total messages overhead

(d) Delay

Fig. 5. Simulation results on the UMassDieselNet trace in terms of control overhead, total overhead, and delivery rate.

control overhead and delivery rate. As expected, *SOFA* always delivers more messages than *SOFA(rand)* because the latter uses random bases that result in larger *diff*s.

While having high delivery rate in four of the traces, *SOFA* and *SOFA(rand)* have comparatively low delivery rates in two smaller traces: Imote (Intel) and Imote (Cambridge). In these traces, especially the Imote (Intel) trace, we can observe that all algorithms using social features perform very poorly, which suggests that the assumption of *correlated interaction*, which is critical for social features based forwarding algorithms, does not hold in these traces.

3) Total message overhead: This is the estimated size of data and control messages generated per message in total, assuming that the size of each data message is 256 bytes. Again, we ignore the control overhead of *Similarity* since it is an ideal algorithm. As shown in Figure 4(c), the total overhead of the *SOFA* or *SOFA(rand)* amounts to no more 50% of that of *Delegation* in most of the traces.

When compare Figure 4(a) with Figure 4(c), we can find that the control overhead accounts for most of the total overhead, especially for forwarding algorithms that use pair-wise routing information, such as *Delegation*. We can also see that the forwarding algorithms using social similarities, including *Social*, *Similarity*, *SOFA*, and *SOFA(rand)*, produce more data overhead than *Delegation*. This is bcause *Delegation* uses pair-wise contact-frequency, which is often the most accurate forwarding metric for message forwarding.

Comparing Figure 4(b) with Figure 4(c), we find that, in several traces, *SprayWait* has a higher forwarding rate and a lower total overhead than *Social*, which is because social similarity can sometimes be misleading. For instance, a group of people with frequent intra-group contact may be mistakenly identified as nodes of high social centrality.

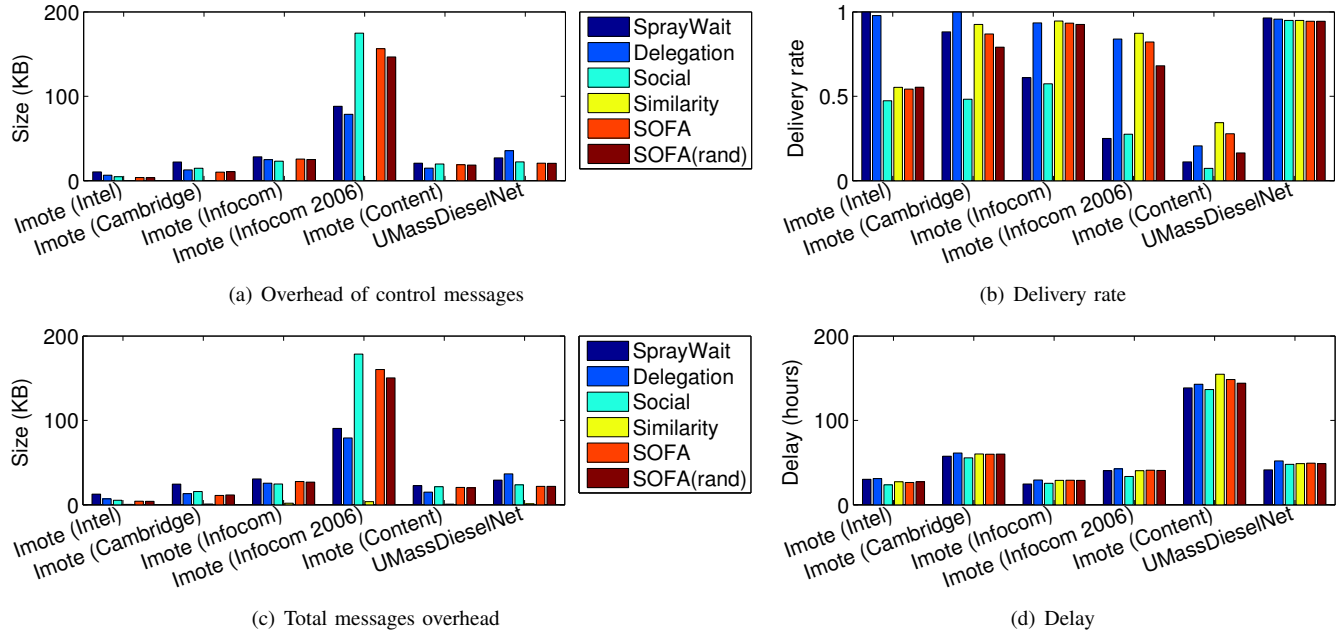
4) Delay: The delay of a message is the time when the first copy of the message reaches the destination. Delays are only measured among delivered messages, since messages that are not delivered do not have this property. The delay for all of the compared algorithms on all traces are similar.

5) Results with various time-to-lives: The performances of the forwarding algorithms are compared in Figure 5, assuming different time-to-lives are used. As shown in Figure 5(b), the delivery rates of all forwarding algorithms increase as time-to-live increases. As shown in Figures 5(a) and 5(c), both the control overhead and the total overhead increase as time-to-live increases. This is because the forwarding algorithms keep encountering nodes and sending control messages, which makes the control overhead per message increase over time. Comparing Figures 5(b) and 5(c), one can see that, even when the forwarding algorithms hardly forward data message any more, the total amount of overheads keep increasing due to the continuously generated control overhead. This suggests another way to reduce control overhead in practice. Finally, all delays increase as time-to-live increases because the messages delivered later have larger delays.

6) Using summary vectors: The results in Figure 6 show that using summary does not result in significant improvements in the delivery rates, but it results in prohibitive control overheads for all algorithms. The increase is especially prominent in the traces that contain large amount of contacts, such as Imote (Infocom 2006), and in algorithms that forwards large number of copies per message in the trace, e.g. *Social*. We conclude that direct use of summary vectors should be avoided.

C. Summary of simulation

Simulation results show that the proposed *SOFA* algorithms have smaller control overhead and comparable delivery rate. In most of the traces, especially the traces with large number of nodes and contacts, *SOFA* provides an interesting trade-off between delivery rate and total message overhead. We conclude that the forwarding performance of *SOFA* is competitive.



(a) Overhead of control messages

(b) Delivery rate

(c) Total messages overhead

(d) Delay

Fig. 6. Results for the algorithms when using summary vectors, which show that the use of summary vectors may create enormous overhead.

VII. Conclusion and Future Work

While existing work focuses on DTN opportunistic forwarding algorithms that increase the delivery rate or decrease data overhead, little attention has been drawn to the control overhead. We proposed using LSH to represent each node as a hash-code, and use these hash-codes to compute the pair-wise similarity that guides forwarding decisions. We introduced three hash-code generation algorithms for three application scenarios. Extensive simulations are performed to evaluate the SOFA algorithms. Future work includes fine-tuning the parameters in the LSH algorithms, such as hash-code length, and applying SOFA on traces where nodes have other types of features, such hotspot contact frequencies.

References

[1] S. Jain, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In *Proc. of ACM SIGCOMM*, 2004.

[2] M. Motani, V. Srinivasan, and P. Nuggenhalli. PeopleNet: Engineering a wireless virtual social network. In *Proc. of ACM MobiCom*, 2005.

[3] A. Chaintreauy, P. Hu, J. Crowcroft, C. Dioty, R. Gassy, and J. Scotty. Pocket Switched Networks: Real-world mobility and its consequences for opportunistic forwarding. In *Tech. Rep. UCAM-CL-TR-617*, 2005.

[4] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networking. In *Proc. of IEEE INFOCOM*, 2006.

[5] Y. Gu and T. He. Data Forwarding in Extremely Low Duty-cycle Sensor Networks with Unreliable Communication Links. In *Proc. of ACM SenSys*, 2007.

[6] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In *Proc. of ACM WDTN*, 2005.

[7] A. Lindgren, A. Doria, and O. Schelen. Probabilistic Routing in Intermittently Connected Networks. *Lecture Notes in Computer Science*, 3126:239–254, August 2004.

[8] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and Focus: Efficient Mobility-Assisted Routing for Heterogeneous and Correlated Mobility. In *Proc. of IEEE PerCom*, 2007.

[9] A. Balasubramanian, B. N. Levine, and A. Venkataramani. DTN Routing as a Resource Allocation Problem. In *Proc. ACM SIGCOMM*, 2007.

[10] H. Dubois-Ferriere, M. Grossglauser, and M. Vetterli. Age Matters: Efficient Route Discovery in Mobile Ad Hoc Networks Using Encounter Ages. In *Proc. of ACM MobiHoc*, 2003.

[11] E. Daly and M. Haahr. Social Network Analysis for Routing in Disconnected Delay-Tolerant MANETs. In *Proc. of ACM MobiHoc*, 2007.

[12] P. Hui, J. Crowcroft, and E. Yoneki. BUBBLE Rap: Social-based Forwarding in Delay Tolerant Networks. In *Proc. of ACM MobiHoc*, 2008.

[13] U. Acer, S. Kalyanaraman, and A. Abouzeid. Weak State Routing for Large Scale Dynamic Networks. In *Proc. of ACM MobiCom*, 2007.

[14] J. Leguay, T. Friedman, and V. Conan. DTN Routing in a Mobility Pattern Space. In *Proc. of ACM WDTN*, 2005.

[15] J. Wu, M. Xiao, and L. Huang. Homing Spread: Community Home-based Multi-copy Routing in Mobile Social Networks. In *Proc. of IEEE INFOCOM*, 2013.

[16] M. Xiao, J. Wu, C. Liu, and L. Huang. TOUR: Time-sensitive Opportunistic Utility-based Routing in Delay Tolerant Networks. In *Proc. of IEEE INFOCOM*, 2013.

[17] A. Vahdat and D. Becker. Epidemic Routing for Partially-connected Ad Hoc Networks. In *Technical Report, Duke University*, 2002.

[18] V. Erramilli, M. Crovella, A. Chaintreau, and C. Diot. Delegation Forwarding. In *Proc. of ACM MobiHoc*, 2008.

[19] D. Gunawardena, T. Karagiannis, A. Proutiere, E. Santos-Neto, and M. Vojnovic. Scoop: Decentralized and Opportunistic Multicasting of Information Streams. In *Proc. of ACM MobiCom*, 2012.

[20] J. Wu and Y. Wang. Social Feature-based Multi-path Routing in Delay Tolerant Networks. In *Proc. of IEEE INFOCOM*, 2012.

[21] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proc. of ACM STOC*, 1998.

[22] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau. CRAWDAD data set cambridge/haggle (v. 2006-09-15). http://crawdad.cs.dartmouth.edu/cambridge/haggle, September 2006.

[23] N. Ristanovic, G. Theodorakopoulos, and J. Le Boudec. Traps and Pitfalls of Using Contact Traces in Performance Studies of Opportunistic Networks. In *Proc. of IEEE INFOCOM*, 2012.