SMART: Lightweight Distributed Social Map Based Routing in Delay Tolerant Networks

Kang Chen and Haiying Shen
Department of Electrical and Computer Engineering
Clemson University, Clemson, SC 29631
Email: {kangc, shenh}@clemson.edu

***Abstract*—Previous Delay Tolerant Network (DTN) routing algorithms exploit either past encounter records (probabilistic routing) or social network properties (social network based routing) to derive a node's probability of delivering packets to their destinations. However, they only have a local view of the network, which limits the routing efficiency. Also, when two nodes meet, they have to exchange the delivery probabilities to the destinations of all packets in the two nodes, which incurs high resource consumption. In a social network, the people a person frequently meets are usually stable, which makes them play a more important role in forwarding message for the person. Based on this, we propose a lightweight distributed Social MAp based Routing algorithm in delay Tolerant networks (SMART). In SMART, each node builds its own social map consisting of nodes it has met and their frequently encountered nodes in a distributed manner. Based on both encountering frequency and social closeness of the two linked nodes in the social map, we decide the weight of each link to reflect the packet delivery probability between the two nodes. The social map enables more accurate forwarder selection through a broader view. Moreover, nodes exchange much less information for social map update and need fewer updates due to social map stability, which reduces resource consumption. Trace-driven experiments and tests on the GENI ORBIT testbed demonstrate the high efficiency of SMART in comparison with previous algorithms.**

I. Introduction

In recent years, the development of wireless networks has stimulated significant research on Delay Tolerant Networks (DTNs) [1]. Among many types of DTNs, we are particularly interested in those consisting of nodes carried by human beings (i.e., pocket switch network [2]) in a specific area (e.g., campus, event site, etc.), due to their ability to support various intriguing applications. For example, students on a campus can share files through mobile phones directly; drivers can obtain traffic and road information from passing vehicles. In this paper, we study the packet routing in DTNs, which is a key function supporting these possible applications. However, it is non-trivial task since mobile nodes meet intermittently and have limited communication ranges and resources in DTNs. Though wireless Internet is common nowadays, we assume a pure DTN scenario without Internet connection in order to better exploit the dynamic routing opportunities in DTNs.

Epidemic routing [3] is a simple way to realize effective routing in DTNs. In this algorithm, when two nodes meet, they exchange the information about all packets and replicate packets that are not on its memory from the other node. As a result, it requires high storage and transmission resources and thus is not practical in DTNs. Other previous routing algorithms in DTNs can be classified into two categories: probabilistic routing [4]–[7] and social network based routing [8]–[12].

978-1-4673-2447-2/12/$31.00 ©2012 IEEE

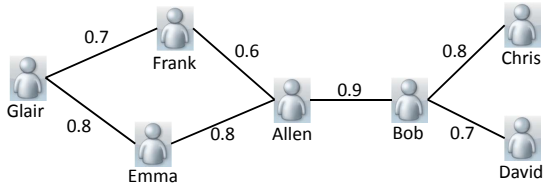


Fig. 1. The social map of Bob.

Probabilistic routing algorithms predict a node's probability of delivering the packet to its destination based on its past encountering records. Packets are always forwarded to nodes with higher delivery probability. Though these algorithms avoid the flooding in epidemic routing, they suffer from two problems. First, the delivery probability is decided by either direct encounter probability or 2-hop accumulated relay probability. Such limited local view in forwarder selection may miss better forwarding opportunities with longer paths. We elaborate the reasons for this drawback in Section III-A. Second, two encountered nodes need to exchange their delivery probabilities to the destination nodes of all packets they carry to decide which packets should be forwarded, which is a non-trivial burden for resource-limited DTNs.

Since mobile device carriers (i.e. human beings) usually are connected with certain social relationships, social network based routing algorithms are proposed recently. They group nodes with frequent contact into communities [8], [9], [11] and/or choose a node with high centrality (i.e., more contacts) or similarity (interest/context/common friends) with the destination node as the packet forwarder [10]–[12]. Essentially, these methods are similar to the probabilistic routing except that they further consider social factors in delivery probability calculation. Therefore, they suffer the same problems as the probabilistic routing methods. In a nutshell, none of them builds the social structure/map on mobile nodes. Therefore, a node only has a view of its own connections/relationships with other nodes or communities but has no knowledge about the surrounding social structure, leading to a restricted local view and limited routing efficiency. Two encountering nodes also need to exchange delivery probabilities, resulting in a high communication overhead.

In order to overcome these shortcomings, we propose a

lightweight distributed Social MAp based Routing algorithm in delay Tolerant networks (SMART). In SMART, each node builds *social map* to record its surrounding social network in DTN. The social map is constructed by learning each encountered node's most frequently met nodes (i.e., stable friends). Each link in the social map is associated with a weight based on the encountering frequency and social closeness of the two connected nodes. The weight is used to deduce the delivery probabilities among nodes. Figure 1 shows an example of social map of Bob. The social map is not limited to one or two hops and reflects possible long relay paths to provide better forwarder selection. When two nodes meet, they only need to exchange the information of their most frequently encountered nodes for social map update. For example, when Bob meets Allen, without querying Allen's probabilities to meet other nodes, he would know that packets for Emma, Frank or Glair should be forwarded to Allen. Also, the stability of most frequently encountered nodes requires no frequent social map update, which reduces resource consumption.

The design of SMART is inspired by a social network property that the people a person frequently meets are usually stable, which makes them play a more important role in message forwarding for the person [13]. For example, we often meet the same colleagues, friends, and family members daily. Our analysis on trace data from the MIT Reality project [14] and the Haggle project [15] in this paper has confirmed this property. SMART does not require social maps to be identical in all nodes or to be complete by including all two-hop nodes in the social structure, which makes the social map construction simple and suitable for distributed DTNs. In summary, our contributions are twofold:

- First, we propose a lightweight distributed social map construction algorithm to enable each node to discover its surrounding social network. To the best of our knowledge, this work is the first to build social maps on individual nodes for DTN routing.
- Second, we propose a new DTN routing algorithm based on the social maps with low cost and high efficiency. It only needs each node's best knowledge of the surrounding network for forwarder selection.

The remainder of this paper is arranged as follows. Related work is introduced in Section II. Section III presents the detailed design of SMART. In Section IV and Section V, the performance of SMART is evaluated through trace-driven experiments and real testbed tests. Section VI concludes this paper with remarks on future work.

II. Related Work

Epidemic routing [3] is a simple DTN routing algorithm, in which two encountering nodes replicate all packets that it hasn't seen from the other node. Due to its flooding nature, this method generates a high efficiency but also a high communication and storage resource consumption. Therefore, it is not suitable for resource-limited DTNs.

Probabilistic routing algorithms [4]–[7] exploit nodes' past encounter records to predict future delivery possibility. In PROPHET [4], the delivery probability considers both direct encountering probability and indirect relay through another node, and is updated upon each encounter and aged over time. Each packet is forwarded to the node with a higher delivery probability. MaxProp [5], RAPID [6], and MaxContribution [7] are similar to PROPHET but further specify the forwarding or storage priorities of different packets based on their delivery probabilities. Packets with higher priority are forwarded first and can replace packets with lower priority. Both RAPID and MaxContribution propose different priority calculation methods that can realize different performance goals such as minimal delay and maximal hit rate. However, the delivery probabilities in these methods only consider at most two-hop relay, which may miss long but effective relay paths, limiting routing efficiency. Also, these methods require two encountering nodes to exchange delivery probabilities to all destinations, which is a heavy burden for mobile nodes.

Social network based DTN routing algorithms [8]–[12] exploit social network properties in DTNs to make forwarding decisions. MOPS [8] builds a publish-subscribe system that groups frequently encountered nodes into groups to facilitate intra-community communication and selects nodes that visit foreign communities frequently for inter-community communication. BUBBLE [9] assigns each node two ranks: global and local. The global rank guides a packet to the community that contains its destination, and the local rank helps to route the packet to its destination within the community. The work in [10] ranks the suitability of a node for carrying a packet based on its centrality and similarity to the packet's destination node. Packets are forwarded from low-rank nodes to high-rank nodes to gradually approach their destinations. The publish-subscribe system in [11] forwards messages to nodes with high utility value, which is calculated based on a node's frequency of encountering subscribers to the interest category of the message and its connectivity with other nodes. HiBop [12] labels each node with various contexts such as personal information, residence, work, and so on. It selects packet forwarder according to nodes' historical encounter records with the context of the packet destinations.

We see that packets in these social network based methods are forwarded to high-rank nodes or nodes in the same community/context with the destinations since such nodes usually have high probability of meeting the destinations. Thus, these methods are similar to the probabilistic routing except incorporating social factors into the calculation of delivery probability. Therefore, they also suffer from the same problems (i.e., local view and high cost) as probabilistic routing. SMART provides each node a broader view of surrounding nodes' encountering activities, hence helps to find a more efficient routing path. It also reduces the resource consumption by reducing the amount of exchanged meeting probability information upon encountering and the social map updates.

III. Algorithm Design

A. The Necessity of Social Map on Routing Efficiency

We first discuss the necessity of social map from the perspective of routing efficiency with a simple scenario shown

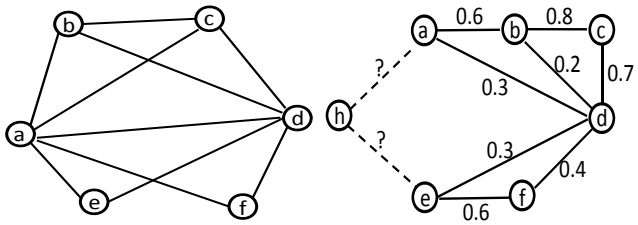
in Figure 2(a). We denote the meeting probability and delivery probability between two nodes as P_{ij} and D_{ij} ($i, j \in \{a, b, c, d, e, f\}$), respectively. The former is the probability of delivering a message to another node directly. The latter refers to the probability of delivering a message to another node through either direct forwarding or indirect relay. We assume d is the destination node.

1) Necessity of Social map: In the routing algorithms that use delivery probability, when a meets b, it updates its delivery probability to d (D_{ad}) by considering the relay through b. In PROPHET [4], $D_{ad} = D_{ad} + (1 - D_{ad}) * P_{ab} * P_{bd} * \beta$, in which $\beta \in [0, 1]$ is a scaling constant. Though the two-hop relay delivery probability provides a wider view than the one-hop meeting probability, the view is still very limited and may miss a faster route with a longer path.

One may claim that using transitive probability calculation can provide much wider view. That is, using b's delivery probability to d (D_{bd}) to update D_{ad}: $D_{ad} = D_{ad} + (1 - D_{ad}) * P_{ab} * D_{bd} * \beta$. Since D_{bd} is already calculated based on all routes from b to d (e.g., $b \rightarrow c \rightarrow d$), the updated D_{ad} can reflect routes more than two hops (e.g., $a \rightarrow b \rightarrow c \rightarrow d$). However, this may lead to delivery probability calculated for a routing path with loops (e.g., $a \rightarrow b \rightarrow a \rightarrow e \rightarrow c \rightarrow d$). We see from the equation that D_{ad} is updated by D_{bd}. But similarly, D_{bd} may be updated by D_{ad} previously, which means D_{bd} has already considered relaying through a. Therefore, by updating with D_{bd}, D_{ad} integrates the relay though itself. In other words, D_{bd} and D_{ad} are boosting each other repeatedly, leading to inaccurate delivery probability.

We confirm this problem with real traces from the MIT Reality project [14] and the Haggle project [15]. The former was obtained from students in the MIT campus, while the latter was collected from 98 scholars attending Infocom'06. Both traces include encountering records among people. We set β to 0.5 and measured the delivery probabilities of all nodes to a randomly selected node using P_{bd} and D_{bd}, respectively. The average delivery probabilities of all nodes are 0.43 and 0.70 in the MIT Reality trace, respectively, and are 0.2 and 0.42 in the Haggle trace, respectively. We see that by replacing P_{bd} with D_{bd}, the delivery probability is exaggerated greatly, thereby cannot provide accurate forwarder selection guidance. Thus, using the transitive probability calculation to enlarge a node's view is not feasible. We propose social map for this purpose without compromising the accuracy of forwarder selection.

2) Benefits of Social Map: Social map avoids the aforementioned loop problem by allowing each node to only record and update its direct encountering probabilities with other nodes. It also provides routes to the destination node with any lengths, thus providing more accurate routing guidance. Figure 2(b) gives a simple scenario to demonstrate this point, in which the number on each link represents the meeting probability between the two connected nodes. Suppose each node has met other nodes sufficiently to learn their meeting probabilities. Node h needs to select a node from a and e as the next hop for a packet towards node d. Without social map, PROPHET cannot consider relay routes that are more than two hops. Then, D_{ad} is $0.3 + 0.6 * 0.2 = 0.42$ and



(a) A small network. (b) A route selection example.

Fig. 2. Network scenarios to show the benefits of social map.

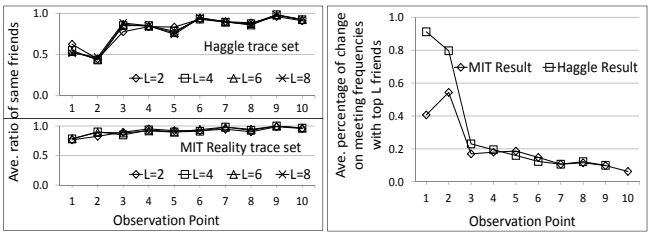
D_{ed} is $0.3 + 0.6 * 0.4 = 0.54$, and node e is a better forwarder than node a. With social map, we can calculate D_{ad} in longer routes (i.e., $a \rightarrow b \rightarrow c \rightarrow d$). Then, D_{ad} is $0.3 + 0.6 * 0.2 + 0.6 * 0.8 * 0.7 = 0.756$, which is larger than D_{ed}, indicating that a is a better forwarder. Therefore, the social map better reflects a node's delivery probability.

B. Social Map Construction

Ideally, the social map should include all nodes in the system. However, this would consume extensive resources for information exchange and storage, thereby should be avoided. Also, the social map structure should be stable to reduce the necessity of timely update, which is hard to realize in DTNs. Moreover, the derived social link weight should be able to reflect actual delivery possibility between the connected nodes for efficient routing. These problems pose two formidable challenges: *i) how can we build stable social maps with a low maintenance cost? ii) How can we define the link weight that can accurately reflect delivery probability?* We introduce our solutions to these challenges in below.

1) Lightweight Social Map Construction: In a social network, a person usually meets his/her major social relations frequently, and they play a more important role in his message forwarding [13]. For example, we meet the same colleagues, friends, and family members daily. Inspired by this, we only keep the nodes a node has met and their top L most frequently encountered nodes (called **top L friends**) in its social map. L can be a fixed value or the number of encountered nodes whose meeting frequencies with the node is higher than a predefined threshold. Due to the stability of a node's top L friends, the social map requires low cost for the structure maintenance and meeting probability update. In following, we first show the stability of top L friends and then introduce the social map construction process and the resulted cost saving.

a) Stability of Top L Friends: In order to verify the stability of a node's top L friends and frequencies of meeting them, we analyzed two traces from the MIT Reality project [14] and the Haggle project [15]. We set ten observation time points evenly in the two traces. At each observation point, we generated the top L friend lists of each node and calculated the ratio of the same top L friends as $\frac{|F_i \cap F_{i+1}|}{|F_i|}$, in which F_i and F_{i+1} denote the set of top L friends at observation point i and $i + 1$, respectively. We measured the ratio when L equal to 2, 4, 6, and 8. The average ratios of all nodes are shown in Figure 3(a). We can see that after the initial two observation points, the average ratio remains very high (around 90%). Note that the length between two observation points is quite long



(a) Evolution of top L friends. (b) Evolution of the change of meeting frequency with top L friends.

Fig. 3. Evolution on the change of friend list and meeting frequency.

in the experiment. This result confirms that a node's most frequently met nodes are very stable.

We further measured the variance of each node's meeting frequencies with its top L friends over time. The frequency change of a node to its friend is measured by $\frac{|f_{i+1}-f_i|}{f_i}$, in which f_{i+1} and f_i denote the meeting frequency with the friend at observation point i and $i+1$, respectively. Figure 3(b) shows the average of all nodes' frequency changes when L equals 4. We see that in both traces, the frequency change is large only at the beginning and decreases to less than 20% after the first two observation points and finally reaches about 10%. This result verifies that a node's meeting frequencies with its top L friends are also relatively stable.

b) Social Map Construction Process: We first introduce a concept of *friendship rank* for a node's top L friends, which represent their meeting frequencies with the node. We divide high meeting frequencies in the system to a number of ranges and assign a rank to each of a node's top L friends based on its meeting frequency with the node. Higher rank represents higher meeting frequency and rank 1 is the highest. As observed from Figure 3(b), a node's meeting frequencies with its top L friends are relatively stable. Therefore, the ranks of a node's top L friends are relatively stable, as shown in Figure 3(a). In this way, we can reduce the cost in social map updates due to the fluctuation of meeting frequencies since a friend's rank does not need to be updated if its meeting frequency changes are within current range. To determine the ranks, we first collect the meeting frequencies of all pairs of nodes for a period of time that is long enough to reflect movement patterns. Then, we evenly split the top 40% high meeting frequencies into l levels, that is, the number of frequencies in each range is the same. The l value and associated ranges can also be pre-defined by the system administrator.

We define each node's top L friends and their friendship ranks as its *friend map*. When two nodes meet, they exchange and update their friend maps. Each node maintains a *social table* that records friend maps of all nodes it has met, as shown in Table I. A node's social map is constructed by connecting all nodes in its social table, with each node only appearing once. A directional link from node i to node j means node j is in the top L friend list of node i. Figure 4(a) shows an example of the social map of node a with L=4.

The social map on a node, say node a, is updated after each encountering with another node rather than at a specific time spot. Specifically, when node a meets node b and receives its friend map, if b is already in a's social map, node a updates b's

TABLE I
SOCIAL TABLE

Node	Top L friends	Friendship ranks
a	f, e, d, g	1, 2, 3, 4
g	d, a, c, h	3, 4 ,4, 5
b	h, c a, j	2, 2, 3, 5
k	i, o, m, n	1, 1, 3, 4
...	...	...

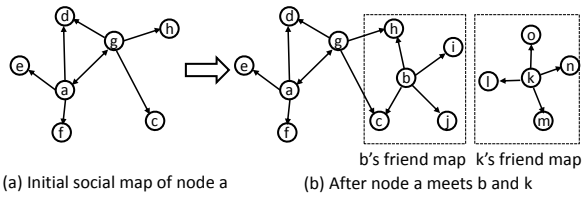


Fig. 4. Social map update process.

L connected nodes in the social map accordingly. Otherwise, node a integrates b's friend map into its social map. Figure 4 demonstrates the update process of node a's social map and Figure 4(a) shows its initial social map. When a meets b, it learns b's top L friends (c, h, i, and j). As h and c are already in the map, node a only adds b, i, and j to its social map. There is no partition in the network. Later, node a meets node k, whose friend map contains l, m, n, and o. Since none of them are in a's social map, a partition is created after they are added into a's social map, as shown in Figure 4(b). In this case, we still regard it as part of the social map because 1) the partition shows some information of the network (i.e., o, l, m, n are good relays for node k), and 2) nodes that can connect partitions may be encountered and inserted into the social map later. Note that though the social map on each node is updated upon each encountering, this process does not consume too much resource since a node's top L friends usually are stable, as shown in Figure 3(a).

Clearly, the value of L affects the social map on each node and the routing efficiency. Since the goal of the social map is to reflect stable social relationships, we can determine L based on the average number of stable friends each node has. This can be realized in both centralized and distributed manner. The former collects the friend information of all nodes to determine an appropriate L, while the latter dynamically adjusts L based on whether stable friends are included in the social map. Due to page limit, we leave this as our future work.

Above algorithm finally generates social maps that are not identical in all nodes and do not include all two-hop nodes. This is similar to our daily lives that each person has his/her own knowledge of the social structure. We will see that the routing efficiency can still be ensured later in Section IV.

Some may question that the social map may fail to reflect some forwarding opportunities, especially for active nodes, since each node can only have at most L links in the social map. We argue that it does not sacrifice the routing performance because 1) the top L friends reflect the major social relationships of each node, which usually take the major roles in message forwarding, and 2) active nodes would appear in more nodes' social maps, thus having more total links for message forwarding. Therefore, the specified L will not compromise routing performance, which is verified in our experimental results in Section IV.

c) Cost Saving Resulted from Social Map: In SMART, two encountered nodes only exchange their top L friends and associated friendship ranks for social map construction. Assume the information of one friend is T bytes, the total amount of data exchanged is about $2TL$ bytes. In previous methods, two encountered nodes exchange their delivery probabilities to the destinations of all packets to make forwarding decision. The size of each delivery probability can be roughly regarded as T bytes too since it also represents the information of a node. We assume packets on each node have N different destinations in average. Then, the total amount of data exchange is about $2TN$ bytes. As a result, the total cost saving is $2T(N-L)M$, where M is the total number of encounters. Recall that M is very large, N is close to the total number of nodes (i.e, hundreds), and L is less than 10. Therefore, the social map greatly reduces the information exchange cost.

2) Social Link Weight Calculation: We assign a weight to the link connecting two nodes, say node i and node j, in a social map to represent the probability of node i successfully forwarding a packet to node j (delivery probability). We consider two factors to calculate the probability: the meeting frequency and the social closeness between the two nodes, which are reflected by friendship rank and shared top L friends [10], respectively. We consider social closeness for weight calculation because people with close relationships are likely to share the same group of friends [10]. A shared top L friend of two nodes is a good relay to forward messages between them since both of them meet the friend frequently. Then, the resultant link weight can more accurately reflect the probability that a packet can be forwarded between the two connected nodes.

We call the link path directly connecting two nodes as 1-hop route and the link path connecting two nodes through one shared top L friend as 2-hop route. The weight of a l-hop ($l = 1$ *or* 2) route between node a and node b, denoted by w_{ab}, is defined as 1 over the sum of the friendship rank of each link in the route:

$$w_{ab} = 1/\sum_{k=0}^{l-1} r_k \tag{1}$$

where r_k denotes the k^{th} link's rank. For two nodes, the weight of one-hop route reflects their meeting frequency while the two-hop routes show the social closeness.

Then, the weight for a link in the social map connecting node a and node b, denoted W_{ab}, integrates all one-hop and two-hop routes between them.

$$W_{ab} = 1/\sum_{k=0}^{m-1} w_{ab_k}, \tag{2}$$

where m is the total number of routes and w_{ab_k} is the weight of the k^{th} route. With this design, the smaller W_{ab}, the higher forwarding probability the two nodes have.

Figure 5 shows an example of part of the social map created on node h, in which L equals 4. We briefly introduce how to calculate the weights of link W_{ab} and W_{ef}. There are three routes between a and b: one one-hop route ($b - a$) and two

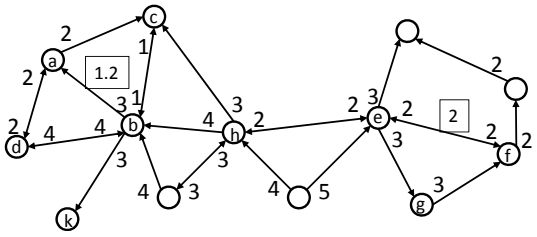


Fig. 5. Part of node h's social map.

two-hop routes through shared top L friend d and c ($a-c-b$ and $a-d-b$). Based on Equation 1, these routes' weights are 1/3, 1/3, 1/6, respectively. Based on Equation 2, $W_{ab} = 1/(\frac{1}{3}+\frac{1}{3}+\frac{1}{6}) = 1.2$. As for e and f, they have only one route: $e-f$. Route $e-g-f$ is not a two-hop route since g is not a shared top L friend, as it only exists in the top L friend list of e. Therefore, $W_{ef} = 1/\frac{1}{2} = 2$.

C. Social Map Based Routing Algorithm

In this section, we first introduce how to calculate the delivery probability between two nodes from link weights and then present the detailed routing process.

1) Delivery Probability: With above social map construction algorithm, the calculated link weight integrates both the meeting frequency and social closeness and thereby can effectively reflect the delivery probability between two nodes. For two directly connected nodes, their link weight can be regarded as the delivery probability between them. For two indirectly connected nodes, we measure the delivery probability between them by the minimal weight of all paths connecting them. The weight of a path is defined as the sum of the weights of the links in the path. Such a definition is based on the fact that when a person tries to send a message to an unfamiliar person, he would usually refer to people having the closest social relationship with him and the unfamiliar person, despite there are many people (i.e., paths) available for reference. Another reason is that the weight of a link connecting two nodes has already integrated the relaying possibility through shared top L friends, alleviating the necessity to consider it again.

The delivery probability of two nodes is the minimum weight of the shortest path between the two nodes. To find the shortest path between node i and node j in the social map, we use the well-known Dijkstra algorithm [16]. Specifically, we take node i as the root node and calculate its shortest paths to all other nodes iteratively until node j is discovered. Then, the reciprocal of the weight of the shortest path is regarded as the delivery probability between the two nodes. If two nodes are disconnected, we regard the delivery probability between them as the minimal possible value.

2) Routing Process: The guideline of our routing algorithm is to always forward a packet to a node whose shortest path to the destination node has the lowest weight. However, there are some issues that need to be addressed, such as incomplete social map, loop prevention and packet replacement strategy. We first discuss approaches to solve these problems and then introduce the routing algorithm.

a) Incomplete Social Map: As stated previously, the social map created on each node may only cover part of the entire

social network. Thus, the destination node of a packet may not exist in the social map. In this case, we rank a node's suitability of forwarding a packet by its active degree, which is measured by the number of links connected to it. Then, the packet is forwarded to the node with the highest active degree until it arrives at a node whose social map contains the destination node. This is inspired by the social network property that an active person can meet more people and thus has a higher probability of meeting the destination [13].

b) Loop Prevention: Since each node maintains its social map independently, forwarding loops may happen in the system. For example, two nodes may believe that the other side is a better forwarder for a packet and forward the packet back and forth repeatedly. To prevent such a loop, we require each packet records the IDs of all nodes that it has been forwarded to. Then, the loop can be avoided by simply forbidding a node to forward a packet to a node that it has visited before.

c) Packet Replacement: It is possible that a node's storage is full when a packet arrives. In this case, SMART simply drops the packet that has lived for the longest period of time.

We then summarize the routing algorithm in SMART as below, with its pseudo-code shown in Algorithm 1.

(1) When two nodes meet each other, they first exchange their friend maps, which are then used to update their social maps (line 2-6). After this, each node processes its packets sequentially.
(2) For the current packet, the node first checks if it has been forwarded to the other node before. If not, it proceeds to step (3). Otherwise, it goes to step (5) (line 10).
(3) By referring to its social map, the node checks whether the other node's shortest path to the packet's destination node has lower path weight than itself. If the destination node is absent from its social map, the node checks whether the other node has higher active degree. If yes, the processing proceeds to step (4). Otherwise, the processing goes to step (5) (line 11-19).
(4) The node forwards the packet to the other node. When the other node receives the packet, if the storage is full, it drops the packet that has been lived for the longest period of time. If the node stores the packet in its memory, it inserts its ID into the packet (line 21 and line 25).
(5) The process of the current packet stops. If there are unprocessed packets, the checking process repeats from step (2) for the next packet (line 8).

In summary, two encountering nodes in SMART only exchange a small amount of information for social map construction and forwarder selection. Also, the delivery probability in SMART naturally considers the multi-hop relay through the top L friends. This global view based forwarder selection can enable more efficient routing.

D. Scalability Discussion

One may question the scalability of SMART on storage in large networks (i.e., with more than 1,000 nodes) since a node needs to store a lot of friend maps. However, we argue that SMART is scalable in memory usage in two aspects. First, recall that one friend map only contains L IDs and link ranks

Algorithm 1 Pseudo-code of the SMART routing algorithm on node n.

```
1:  Upon the reception of Hello message from node m
2:  procedure exchangeTopFriendsWith(m)
3:      n.sendTopFriendsTo(m)
4:      n.receiveTopFriendsFrom(m)
5:      n.updateSocialMap()
6:  end procedure
7:  procedure exchangePacketsWith(m)
8:      for each packet k in node n do
9:          bForward ← false
10:         if k.hasBeenOn(m) ≠ true then
11:             if n.containInMap(k.des) then
12:                 if m.getW(k.des) < n.getW(k.des) then
13:                     bForward ← true
14:                 end if
15:             else
16:                 if m.getDegree( ) > n.getDegree( ) then
17:                     bForward ← true
18:                 end if
19:             end if
20:             if bForward = true then
21:                 n.forwardPacketTo(k, m)
22:             end if
23:         end if
24:     end for
25:     n.receivePacketsFrom(m)
26: end procedure
```

(only about $8L$ bytes). Then, the storage of 10,000 friend maps requires about $80L$ KB memory, which is not a big burden for most mobile devices nowadays. Second, a node can only store the friend maps of its frequently met nodes since a node usually meet a limited number of nodes frequently, which are forwarders for the node's messages in most cases. This is similar to our daily lives that people usually only remember the information of limited number of friends.

IV. Trace-driven Performance Evaluation

We first conducted event-driven experiments using real traces from the MIT Reality project [14] and the Haggle project [15]. We compared SMART with following representative DTN routing algorithms.

(1) *PROPHET*: PROPHET is a probabilistic routing algorithm. It calculates delivery predictability based on past encountering records and forwards packets to nodes with a higher delivery predictability to destinations.

(2) *SimBet*: SimBet is a social network based algorithm. It calculates the suitability of a node for carrying a packet by the node's centrality value and its similarity (the number of shared encountered nodes) to the destination node. Packets are always forwarded to nodes with better suitability.

(3) *StaticWait*: In StaticWait, a source node carries its packet until meeting the packet's destination. We use this algorithm as a baseline method to show the routing efficiency when no active forwarding strategy is adopted.

In the experiment, the first 1/3 of both traces were used as the initialization period to collect enough encountering records. After this, packets were generated at the rate of R_n per 300s and per 40s in the MIT trace and the Haggle trace, respectively. The size of a packet was set to 1 KB. The source and destination of a packet were randomly selected from all

nodes in the system. In SMART, L was set to 4 by default. In PROPHET and SimBet, the parameters used to calculate the delivery probability and utility were configured the same as in their papers. We used the same packet replacement algorithm as in SMART for PROPHET and SimBet.

We tested the performance of SMART with different packet rates, different memory sizes on each node, and different values of L. In the test with different packet rates, the total number of packets was varied from 5,000 to 25,000 (i.e., R_n was varied from 1 to 5 with an increase of 1 in each step). The memory size on each node was set to 100KB. In the test with different memory sizes, the memory on each node was varied from 60KB to 140 KB with an increase of 20 KB in each step, and the packet rate (R_n) was set to a medium value of 3. In the test with different values of L, we varied the value of L from 2 to 8, and set the packet rate to 3 and memory size on a node to 100 KB.

We measured the following metrics during the test, and the confidence interval was set to 95%.

- *Hit rate:* the percentage of requests that are successfully delivered to their destination nodes at the end of the experiment. This metric represents the effectiveness of successful delivery of a routing method.
- *Average delay:* the average delay time of all successfully delivered packets. This metric represents the efficiency of a routing method.
- *The number of forwarding hops:* the total number of packet forwarding hops throughout the experiment. This metric represents the efficiency of a routing method.
- *Routing cost:* the number of information units exchanged between two encountering nodes throughout the experiment. This metric shows the cost incurred for maintaining necessary information needed for forwarder selection.

A. Performance with Different Packet Generating Rates

1) Hit Rate: Figure 6(a) and Figure 7(a) demonstrate the hit rates of the four methods with the Haggle trace and the MIT Reality trace, respectively. From the two figures, we see that the hit rates of the four methods follow SMART>SimBet>PROPHET>StaticWait.

StaticWait shows the lowest hit rate because packets only statically wait in their generators to reach destinations. By exploiting encountering probabilities between nodes, PROPHET, SimBet, and SMART achieve higher hit rate than StaticWait. SMART deduces nodes' delivery probabilities to destinations relying on relatively stable social maps. It can choose an optimal forwarder in a long path to the destination with a broad view without being limited by the local view of directly encountered nodes, thereby generating the highest hit rate. PROPHET and SimBet evaluate the delivery probability within one or two hops by past encounters, which cannot provide long routing paths, leading to a lower hit rate than SMART.

We see that the hit rate of SimBet is slightly higher than that of PROPHET with the Haggle trace and is overlapped with that of PROPHET with the MIT Reality trace. This is caused by the different properties of the two traces. The Haggle trace was obtained from a conference, in which nodes meet with each other frequently in a small area. Then, nodes with high centrality in the test with the Haggle trace can meet more nodes and have high forwarding ability. By integrating centrality in forwarder selection, SimBet generates slightly higher hit rate than PROPHET. On the other hand, the MIT Reality trace represents a loose campus environment, so the centrality used in SimBet cannot contribute much to forwarding efficiency. Therefore, PROPHET and SimBet have similar performance.

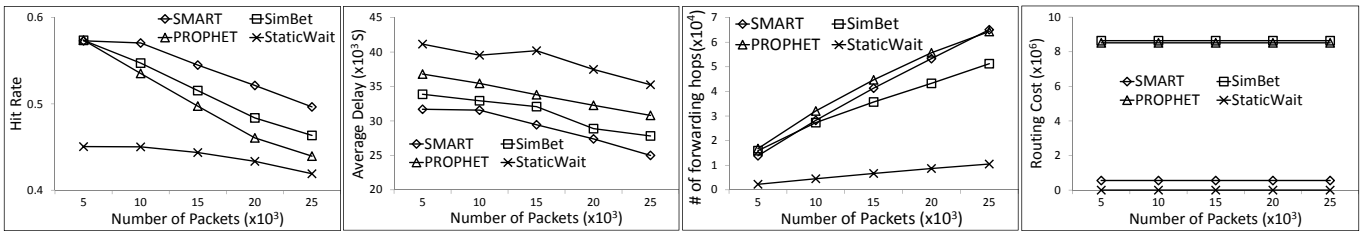
We observe that as the total number of packets increases, the hit rates of the four methods decrease. This is because the forwarding opportunities and memory available for packet forwarding are limited. Then, when more packets are generated, more packets get replaced in nodes, leading to a decreased hit rate. It is interesting to see that the hit rates of the three methods with active forwarding decrease more quickly than that of StaticWait. In these three methods, certain nodes tend to receive more packets since they are more active or connect to more nodes. Then, there are more packet replacements on them. But in StaticWait, packets are relatively evenly distributed among all nodes, which means that the memory on all nodes are utilized, leading to fewer replaced packets.

With above results, we conclude that SMART can achieve efficient routing in the DTN environment with the proposed social map. These results also justify the correctness of the design of the link weight calculation method that considers both meeting frequency and social closeness.

2) Average Delay: Figure 6(b) and Figure 7(b) show the average delays of the four methods in the tests with the Haggle trace and the MIT Reality trace, respectively. From the two figures, we find that the average delays of the four methods follow SMART<SimBet<PROPHET<StaticWait at different packet rates. SMART has the lowest average delay because it considers multi-hop forwarding opportunities when making forwarding decisions with a broader view from the social map, which enables a packet to travel through a fast route to its destination. Both PROPHET and SimBet fail to consider long routing paths that may generate shorter delay than the 1-hop or 2-hop paths. Therefore, they produce higher average delay than SMART. StaticWait has the highest average delay since packets only wait in their initiators for destinations without being forwarded. Such results further demonstrate the high efficiency of SMART in terms of routing delay.

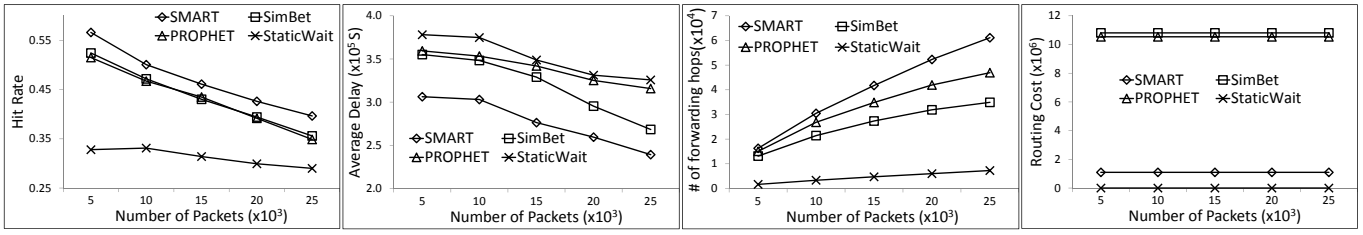
We also find that the average delays of the four methods decrease as the packet generating rate increases. This is caused by two reasons: 1) we only counted the average delay of successfully delivered packets, and 2) the four methods drop the oldest packet when the storage in a node is full upon a packet arrival. As a result, when the number of packets increases in the system, more old packets with the large delays are dropped, leading to a lower average delay.

3) The Number of Forwarding Hops: Figure 6(c) and Figure 7(c) show the number of forwarding hops of the four methods with the Haggle trace and the MIT Reality trace, respectively. We observe that StaticWait and SimBet produce the smallest and the second to the smallest number of forwarding hops, respectively. StaticWait lets each packet wait in its initiator for the destination, so each packet is forwarded



(a) Hit rate. (b) Average delay. (c) Number of forwarding hops. (d) Routing cost.

Fig. 6. Performance of each method with the Haggle trace under different packet rates.



(a) Hit rate. (b) Average delay. (c) Number of forwarding hops. (d) Routing cost.

Fig. 7. Performance of each method with the MIT Reality trace under different packet rates.

at most once. SimBet considers both similarity and centrality. Therefore, the nodes with high centrality (i.e., actively contact many other nodes) afford most of the forwarding load, leading to a small number of forwarding hops. We also see that PROPHET has slightly more forwarding hops than SMART with the Haggle trace and fewer forwarding hops than SMART with the MIT Reality trace. PROPHET always forwards a packet to nodes with high meeting frequencies to the destination. Then, as nodes in the Haggle trace meet each other frequently, it is easy to find a better forwarder in PROPHET even though it is only slightly better, thus generating many packet forwarding hops. But in the MIT Reality trace, meeting frequencies are more diverse and packets are mainly forwarded to few nodes having high meeting frequencies with their destinations, leading to less packet forwarding.

4) Routing Cost: Figure 6(d) and Figure 7(d) plot the routing costs of the four methods with the Haggle trace and the MIT Reality trace, respectively. We see that StaticWait has no routing cost since no information exchange is needed. SMART incurs a significantly lower routing cost than PROPHET and SimBet. This is because two encountered nodes only need to exchange their friend maps with L (usually a small value) entries in SMART, while nodes need to exchange the information regarding the destination nodes of all packets in PROPHET and SimBet. SimBet has a slightly higher routing cost than PROPHET since in addition to the similarity information, a node has to send its centrality information to the newly met node. In a nutshell, StaticWait incurs the least total costs but has a low efficiency, SMART consumes low total transmission and storage costs, and PROPHET and SimBet generate very high total transmission and storage cost. This result confirms SMART's low-cost on information exchange.

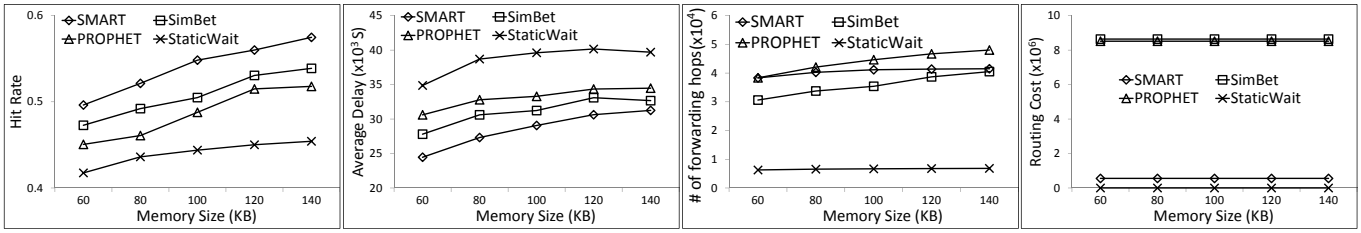
B. Performance with Different Memory Sizes on Each Node

1) Hit Rate: Figure 8(a) and Figure 9(a) demonstrate the hit rates of the four methods with the Haggle trace and the MIT Reality trace when the memory size varies, respectively. We see that the hit rates of the four methods follow the same as in Figure 6(a) and Figure 7(a) due to the same reasons. We also find that when the memory size on a node increases, the hit rates of all methods also increase. This is because with lager memory size, each node can carry and forward more packets to their destinations, leading to a higher hit rate.

2) Average Delay: Figure 8(a) and Figure 9(a) show the average delay of the four methods with the Haggle trace and the MIT Reality trace when the memory size on each node varies, respectively. We find that the average delays also match the results in Figure 6(a) and Figure 7(a) for the same reasons. It is interesting to see that when the memory size on a node increases, the average delays also increase. This is because 1) SMART always replaces the oldest packet with the longest delay when memory is full, and 2) we only counted the delays of successful packets. Therefore, when memory size increases, fewer old packets are dropped, thus producing more successfully delivered packets with longer delays.

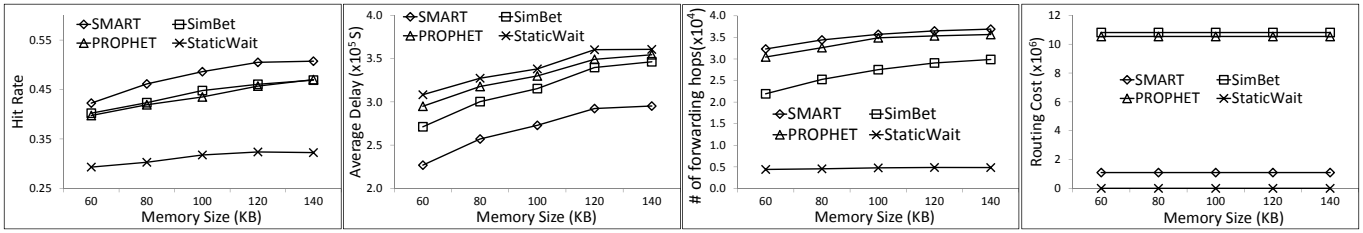
3) The Number of Forwarding Hops: Figure 8(c) and Figure 9(c) show the number of forwarding hops of the four methods with the Haggle trace and the MIT Reality trace when the memory size on each node varies, respectively. We find that the number of forwarding hops follow the same relationship as in Figure 6(c) and Figure 7(c) due to the same reasons. We also see that when the memory size in a node increases, the forwarding hops in the four methods increase slightly. This is because when each node can store more packets, more forwarding operations occur during each node encountering.

4) Routing Cost: Figure 8(d) and Figure 9(d) demonstrate the routing costs of the four methods with the Haggle trace and the MIT Reality trace when the memory size on each node varies, respectively. We find that the routing costs are the same as the Figure 6(d) and Figure 7(d) for the same reasons. This is because the routing cost is irrelevant to the number of packets on each node but only related to the L for SMART and the number of nodes in the system for other three methods. Combining all the above results, we conclude that SMART has superior performance than other methods in DTN routing with different memory sizes on each node.



(a) Hit rate. (b) Average delay. (c) Number of forwarding hops. (d) Routing cost.

Fig. 8. Performance of each method with the Haggle trace under different memory sizes.



(a) Hit rate. (b) Average delay. (c) Number of forwarding hops. (d) Routing cost.

Fig. 9. Performance of each method with the MIT Reality trace under different memory sizes.

C. Effect of the L Value

In this section, we varied the value of L used in SMART from 2 to 8 to evaluate its effect on the routing performance and verify that large L values (i.e., greater than a threshold) would not significantly enhance the routing performance. The total number of packets was set to a medium value of 15,000. The results are shown in Table II and Table III.

TABLE II
ROUTING PERFORMANCE WITH THE HAGGLE TRACE

L	Hit Rate	Ave. Delay (s)	Forwarding Hops	Routing Cost
2	0.541425	30,843.3	40,544	277,704
3	0.551023	29,679.3	40,765	416,556
4	0.551156	29,227.1	41,750	555,408
5	0.560021	29,125.4	41,542	833,112
6	0.56822	29,813	41,427	833,112
7	0.572219	29,675.7	41,419	971,964
8	0.576218	29,790.9	41,272	1,110,816

TABLE III
ROUTING PERFORMANCE WITH THE MIT REALITY TRACE

L	Hit Rate	Ave. Delay (s)	Forwarding Hops	Routing Cost
2	0.431467	278,990	26,074	546,892
3	0.459933	276,096	32,948	820,338
4	0.481467	273,978	35,088	1,093,784
5	0.502867	280,568	36,845	1,367,230
6	0.502533	272,873	37,899	1,640,676
7	0.515667	279,737	38,631	1,914,122
8	0.515933	283,755	38,345	2,187,568

1) Hit Rate: We see that the hit rate of SMART increases steadily when L increases from 2 to 8. This is because the calculation of link weight depends on the number of shared top L friends. Therefore, when L increases, the calculated weight can reflect the delivery probability of the two connected nodes more precisely. Consequently, a node can more correctly decide if a newly met node is a better carrier for its packets, leading to improved routing efficiency. This result confirms the feasibility of constructing a social map by exchanging only the top L friends in DTNs and implies that L can be adjusted to achieve a tradeoff between cost and routing efficiency.

We also see that the incensement of hit rate with the MIT Reality trace is much larger than that with the Haggle trace when L increases. This is because the two traces were obtained in different environments. The Haggle trace was conducted in a crowded conference scenario, in which each node can meet many nodes frequently; so a small L can still approximately represent the social structure and will not decrease the hit rate significantly. However, the MIT Reality trace was obtained in a sparse campus environment. In this case, a small L would lose some important friends, thereby providing fewer forwarding opportunities and leading to a low hit rate.

2) Average Delay: We find that the average delay of SMART decreases when L increases at the beginning and remains at the same level when L is larger than a medium value (i.e., 5). Recall that we only count the delay of successful packets. When the L increases from a small value, as stated in previous section, the social map constructed on each node becomes more and more complete, resulting in better forwarder selection and decreased average delay even when more packets are delivered. After L reaches the medium value, which enables social maps to show almost all most frequently met nodes and optimal path, the enhancement in the routing efficiency is not significant if L further increases, as shown in the first column of the two tables. Therefore, when more packets are successfully delivered, their average delays remain on the same level. As a result, the average delay fluctuates at a certain level when the number of delays increases.

3) The Number of Forwarding Hops: We see that when L increases from 2 to 8, the number of forwarding hops of SMART increases in the test with the MIT Reality trace but remains at the same level when the Haggle trace is used. This is also caused by the different environments of the two traces. In the Haggle project, nodes are more crowded, so it is still easy to find a next-hop node even when L is small. But in the MIT Reality trace, nodes are sparser, so a small L may not be able to reflect the social structure well, leading to fewer forwarding opportunities. This also matches the results about

hit rate in the two tables.

4) Routing Cost: We find that the routing cost increases in proportion to the value of L when it increases from 2 to 8 with both traces. This is because the routing cost is actually the number of encounters multiplied by $2L$. We see that the routing cost is still quite small when L is 8 compared to that of SimBet and PROPHET shown in Figure 6(d) and Figure 7(d). This result shows the efficiency of SMART in reducing information transmission, and also implies that it is important to find an optimal L value that generates low routing cost while achieving high routing efficiency.

5) Summary: Above experimental results indicate that SMART still works efficiently when L is set to a small value. For example, even when $L = 2$, SMART still generates close performance on hit rate, average delay, and cost with other compared methods in Figure (8) and Figure (9). Also, the performance of SMART is improved when L increases and remains stable when L is larger than 5. This result justifies the idea that top L friends can provide a sufficient reflection of the social structure in a DTN and provide critical information to guide packet forwarding. SMART achieves an optimal balance between efficiency and cost when L is set to a medium value such as 4 or 5 for the two traces. This value may change in different scenarios. However, since the increase in the routing cost is linear when L increases, we conclude that SMART is energy efficient and suitable for DTNs.

V. GENI Experiment

We further evaluated the performance of the four methods on the real-world GENI ORBIT testbed [17], [18]. In ORBIT, nodes communicate with nodes within communication range through the wireless interface. Since all nodes are fixed in ORBIT, we again used the MIT Reality trace to drive node encountering. The total number of packets was set to 15,000, and the memory on each node was set to 100 KB.

TABLE IV
Efficiency and cost in the GENI test

Method	Hit Rate	Ave. Delay (s)	Forwarding Cost	Routing Cost
SMART	0.511	251,954.4	41,954	1,103,472
SimBet	0.496	269,007.0	32,357	14,173,600
PROPHET	0.501	271,546.5	24,357	12,315,306
StaticWait	0.31	308,046.1	5,263	0

The test results are shown in Table IV. We see that SMART still produces the highest hit rate, the lowest average delay and the second lowest cost compared to other three methods for the same reasons described previously. Such results further prove the high efficiency of SMART in the real-world testbed.

VI. Conclusion

In this paper, we propose SMART, a lightweight distributed social map based routing algorithm for DTNs. By exploiting the social network properties that a person's most frequently encountered friends often remain stable, SMART enables each node to build a social map recording its knowledge of surrounding social structure. Specifically, nodes exchange the top L most frequently encountered nodes when they meet for social map construction. In the social map, the delivery probability between two nodes is evaluated by considering both meeting frequency and social closeness. Then, packets are forwarded to nodes with higher delivery probability to their destinations. SMART is more efficient than previous probabilistic routing algorithms because the social map offers a broader view for forwarder selection. Moreover, two encountering nodes only need to exchange the information of their top L friends, which is relatively stable, leading to a low information exchange and update overhead. Extensive real-trace driven experiments and testbed tests demonstrate the effectiveness and efficiency of SMART in comparison with previous algorithms. In the future, we plan to investigate the L value that leads to the optimal tradeoff between routing efficiency and cost, and use adaptive L for nodes with different active levels to further improve routing efficiency.

Acknowledgements

This research was supported in part by U.S. NSF grants CNS-1249603, OCI-1064230, CNS-1049947, CNS-1156875, CNS-0917056 and CNS-1057530, CNS-1025652, CNS-0938189, CSR-2008826, CSR-2008827, Microsoft Research Faculty Fellowship 8300751, and U.S. Department of Energy's Oak Ridge National Laboratory including the Extreme Scale Systems Center located at ORNL and DoD 4000111689.

References

[1] S. Jain, K. R. Fall, and R. K. Patra, "Routing in a delay tolerant network," in *Proc. of SIGCOMM*, 2004.
[2] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Pocket switched networks and human mobility in conference environments," in *Proc. of WDTN*, 2005.
[3] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," Duke University, Tech. Rep., 2000.
[4] A. Lindgren, A. Doria, and O. Scheln, "Probabilistic routing in intermittently connected networks." *Mobi. Compt. and Commun, Rev.*, 2003.
[5] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "MaxProp: Routing for vehicle-based disruption-tolerant networks," in *Proc. of INFOCOM*, 2006.
[6] A. Balasubramanian, B. N. Levine, and A. Venkataramani, "DTN routing as a resource allocation problem." in *Proc. of SIGCOMM*, 2007.
[7] K. Lee, Y. Yi, J. Jeong, H. Won, I. Rhee, and S. Chong, "Max-Contribution: On optimal resource allocation in delay tolerant networks." in *Proc. of INFOCOM*, 2010.
[8] F. Li and J. Wu, "MOPS: Providing content-based service in disruption-tolerant networks," in *Proc. of ICDCS*, 2009, pp. 526–533.
[9] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: social-based forwarding in delay tolerant networks," in *Proc. of MobiHoc*, 2008.
[10] E. M. Daly and M. Haahr, "Social network analysis for routing in disconnected delay-tolerant manets," in *Proc. of MobiHoc*, 2007.
[11] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco, "Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks," *IEEE JSAC*, vol. 26, no. 5, pp. 748–760, 2008.
[12] B. Chiara and *et al.*, "Hibop: A history based routing protocol for opportunistic networks," in *Proc. of WoWMoM*, 2007.
[13] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott., "Impact of human mobility on the design of opportunistic forwarding algorithms," in *Proc. of INFOCOM*, 2006.
[14] N. Eagle, A. Pentland, and D. Lazer, "Inferring social network structure using mobile phone data," *PNAS*, vol. 106, no. 36, 2009.
[15] A. Chaintreau, P. Hui, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Impact of human mobility on opportunistic forwarding algorithms," in *Proc. of INFOCOM*, 2006.
[16] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, 1959.
[17] "GENI project," http://www.geni.net/.
[18] "ORBIT," http://www.orbit-lab.org/.