# SMART: Utilizing Distributed Social Map for Lightweight Routing in Delay-Tolerant Networks

Kang Chen, *Student Member, IEEE*, and Haiying Shen, *Senior Member, IEEE, Member, ACM*

***Abstract*—Previous delay-tolerant network (DTN) routing algorithms exploit either past encounter records or social network properties 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 abilities to the destinations of all packets in the two nodes, which incurs high resource consumption. In this paper, we propose SMART, which utilizes a distributed social map for lightweight routing in delay-tolerant networks. 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 ability 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, which reduces resource consumption. Trace-driven experiments and tests on the GENI ORBIT testbed demonstrate the high efficiency of SMART in comparison to previous algorithms.**

***Index Terms*—Delay-tolerant networks, routing, social map.**

## 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 (human-based DTNs) in a specific area (e.g., rural village, campus, and local community) due to their ability to support various applications. For example, messages can be relayed by devices carried by humans to realize data communication in rural areas with no or limited infrastructures. The data can further be gathered to a location with access points to enable intermittent network connection in the whole area [2]. In pocket switch networks (PSNs) [3], digital devices carried by people are dynamically networked, and the encountering-based message forwarding can help discover geo/social relationships among people and provide applications such as friend recommendation, distributed file sharing, or Question&Answer systems in a local community [4]. Though infrastructures exist in some scenarios, like PSNs, we focus on the distributed encountering-based packet routing in DTNs, which is the key function supporting the DTN applications. However, this is a nontrivial task since mobile nodes meet intermittently and have limited communication ranges and resources.

Epidemic routing [5] 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 [6]–[9] and social-network-based routing [10]–[14].

Probabilistic routing algorithms predict a node's probability of delivering the packet to its destination (i.e., delivery ability) based on its past encountering records. Packets are always forwarded to nodes with higher delivery ability. Though these algorithms avoid the flooding in epidemic routing, they suffer from two problems. First, the delivery ability 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 abilities to the destination nodes of all packets they carry to decide which packets should be forwarded, which is a nontrivial 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 have been proposed recently. They group nodes with frequent contact into communities [10], [11], [13] 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 [12]–[14]. Essentially, these methods are similar to the probabilistic routing except that they further consider social factors in delivery ability calculation. Two nodes still exchange information on delivery abilities for different destinations. Therefore, they suffer from the same problems: limited view and high communication overhead on information exchange.

In order to overcome these shortcomings, we propose SMART, which utilizes a distributed social map for lightweight routing in delay-tolerant networks. In SMART, each node builds a *social map* to record its surrounding social network in DTNs. 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 abilities among nodes. Fig. 1 shows an example of the social

Manuscript received December 12, 2012; revised April 02, 2013 and June 16, 2013; accepted August 21, 2013; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor L. Qiu. This work was supported in part by the US NSF under Grants CNS-1254006, CNS-1249603, CNS-1049947, CNS-0917056, and CNS-1025652 and the Microsoft Research Faculty Fellowship 8300751.

The authors are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29631 USA (e-mail: kangc@clemson.edu; shenh@clemson.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2013.2281583

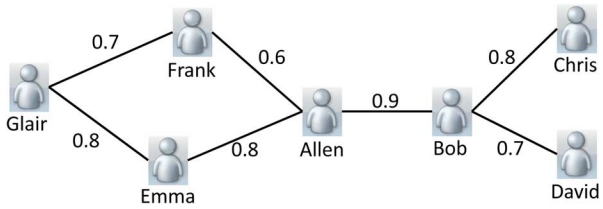



Fig. 1. Social map of Bob.

map of Bob. The social map is not limited to 1 or 2 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 means no frequent social map update is needed, 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 an important role in forwarding packets for the person [15]. For example, we often meet the same colleagues, friends, and family members daily. Our analysis on trace data from the MIT Reality project [16] and the Haggle project [17] also confirms this property. Consequently, SMART applies to scenarios in which node carriers belong to certain social structures and present the above social properties. SMART does not require social maps to be identical in all nodes or to be complete (i.e,, including all nodes), which makes the social map construction simple and suitable for distributed DTNs. In summary, our contributions are threefold.

- 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.
- Third, extensive trace-driven experiments and tests on the GENI ORBIT testbed show the efficiency and effectiveness of SMART in comparison to previous algorithms.

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 Sections IV and 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

In epidemic routing [5], each of the two encountering nodes replicates all packets it has not 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 [6]–[9] exploit nodes' past encounter records to predict future delivery ability. In PROPHET [6], the delivery ability considers both direct encountering probability and indirect relay through another node and is updated upon each encounter and aged over time. A packet is always forwarded to the node with higher delivery ability. MaxProp [7], RAPID [8], and MaxContribution [9] are similar to PROPHET, but further specify the forwarding or storage priorities of different packets based on their delivery abilities. RAPID and MaxContribution propose different priority calculation methods for different goals such as minimal delay and maximal hit rate.

Social-network-based DTN routing algorithms [10]–[14] exploit social network properties in DTNs to make forwarding decisions. MOPS [10] builds a publish–subscribe system that groups frequently encountered nodes to facilitate intracommunity communication and selects nodes that visit foreign communities frequently for intercommunity communication. BUBBLE [11] 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 [12] ranks the suitability of a node for carrying a packet based on its centrality and similarity to the packet's destination node. The publish–subscribe system in [13] 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 [14] labels each node with various contexts such as personal information, residence, work, and so on. It decides packet forwarder according to nodes' historical encounter records with the context of the packet destinations.

We see that all the above methods only differ on ways to deduce a node's delivery ability to a destination. Therefore, they both suffer the issue of local view and high cost. SMART provides each node a broader view of surrounding nodes' frequently met nodes, hence helping to find a more suitable packet forwarder. It also saves the resource consumption by reducing the amount of exchanged information upon node encountering and the frequency of social map update.

## III. Algorithm Design

### A. Benefits of Social Map on Routing Efficiency

We first discuss the benefits of a social map from the perspective of routing efficiency with a simple scenario, shown in Fig. 2(a). We denote the meeting probability and delivery ability 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 upon their encountering. 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) Drawback of Previous Methods:* In routing algorithms that use delivery ability, when $a$ meets $b$, it updates its delivery ability to $d$ $(D_{ad})$ by considering the relay through $b$. In PROPHET [6], $D_{ad} = D_{ad} + (1 - D_{ad}) * P_{ab} * P_{bd} * \beta$, in which $\beta \in [0, 1]$ is a scaling constant. Such updates only consider 2-hop relay delivery ability (i.e., $a \rightarrow b \rightarrow d$), which has limited view on forwarder selection and may miss a forwarder on a faster but longer path.

One may claim that using transitive probability calculation can provide a much wider view. That is, using $b$'s delivery ability 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

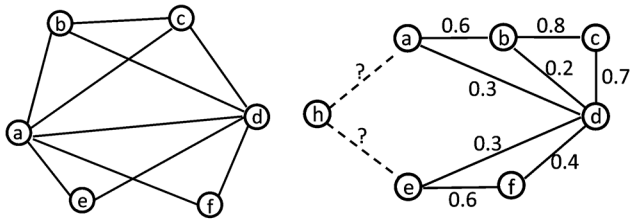



Fig. 2. Network scenario to show the benefits of social map. (a) Small network. (b) Example on route selection.

TABLE I
CHARACTERISTICS OF MOBILITY TRACES

| | MIT Reality | Haggle |
|---|---|---|
| **# Nodes** | 94 | 98 |
| **Location** | Campus | Conference |
| **Duration** | 30 days | 4 days |
| **# Encountering** | 137936 | 74224 |
| **# Encountering per day** | 4597 | 18556 |

based on all routes from $b$ to $d$ (e.g., $b \rightarrow c \rightarrow d$), the updated $D_{ad}$ can reflect routes more than 2 hops (e.g., $a \rightarrow b \rightarrow c \rightarrow d$). However, this may lead to delivery ability 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 $D_{bd}$. However, 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 $D_{ad}$ may boost each other repeatedly, leading to inaccurate delivery ability.

We confirm this problem with real traces from the MIT Reality project [16] and the Haggle project [17]. The former was obtained from students and staffs on the MIT campus, while the latter was collected from 98 scholars attending the IEEE INFOCOM 2006. Both traces include encountering records among people. Table I shows the summary of the two traces.

We set $\beta$ to 0.5 and measured the delivery abilities of all nodes to a randomly selected node using $P_{bd}$ and $D_{bd}$, respectively. The average delivery abilities 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 ability is exaggerated greatly, thereby it cannot provide accurate forwarder selection guidance. Thus, it is not feasible to use the transitive probability to enlarge the view during forwarder selection. We then propose a social map for this purpose with a controllable cost.

*2) Benefits of Social Map:* The social map on a node provides a much broader view naturally. A node can discover routes to the destination with any lengths, thus providing more accurate forwarder selection. Fig. 2(b) gives a simple example, in which the number on each link represents the meeting probability between the two connected nodes. Suppose each node has learned their meeting probabilities. Node $h$ needs to select a node from $a$ and $e$ as the next hop for a packet toward node $d$. Without a social map, PROPHET cannot consider relay routes that are more than 2 hops. Then, since $D_{ad}$ is $0.3+0.6*0.2 = 0.42$ and $D_{ed}$ is $0.3 + 0.6 * 0.4 = 0.54$, node $e$ is a better forwarder than node $a$. With a social map, we can consider longer routes (i.e., $a \rightarrow b \rightarrow c \rightarrow d$) for $D_{ad}$: $0.3+0.6*0.2+0.6*0.8*0.7 = 0.756$, which is larger than $D_{ed}$. Then, node $h$ can select the correct forwarder (i.e., $a$). There are already several ways to compute the weight between two nodes in a graph when the weight of each edge is known [18]–[20]. These methods require the weight of every edge in the network and much calculation for each source–destination pair, which cannot be satisfied in DTNs, i.e., a node cannot know all link weights and the routing process needs to calculate the weights between many pairs of nodes. Therefore, we adopt a different way to calculate the delivery ability between two nodes on the social map, as explained in Section III-C.1.

The social map provides benefits on routing, which are actually resulted from the cost of maintaining more information (i.e., top $L$ friends) on each node. The social map only contains a node's major relationship, which is stable and requires less frequent updates. Therefore, it actually provides an acceptable balance on cost and routing performance.

### 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. Also, the social map structure should be stable to reduce the necessity of timely update, which is hard to realize in DTNs. Moreover, the social link weight should be able to reflect the delivery possibility between connected nodes for efficient routing. These problems pose two challenges: *1) how to build stable social maps with a low maintenance cost; 2) how to define the link weight that can accurately reflect delivery ability*. We introduce our solutions to these challenges in the following.

*1) Lightweight Social Map Construction:* In a social network, a person usually meets his/her major social relations frequently, who play a more important role in his message forwarding [15]. 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 the node's social map. $L$ can be a fixed value or the number of encountered nodes whose meeting frequencies with the node are 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 update. Below, we first show the stability of top $L$ friends and then introduce the social map construction process, the coverage of the resulted social map, the ways to determine the value of $L$, 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 the frequencies of meeting them, we analyzed the MIT Reality project [16] trace and the Haggle project [17] trace. We see from the summary in Table I that nodes move actively in the two traces.

We set 10 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 $|F_i \cap F_{i+1}|/|F_i|$, in which $F_i$ and $F_{i+1}$ denote the set of top $L$ friends at observation point $i$ and $i+1$, respectively. In order to get $F_i$, SMART counts the encounters from the start time to observation point $i$. We measured the ratio when $L$ equal to 2, 4, 6, and 8. The average ratios of all nodes are shown in Fig. 3(a). We can see that after the initial two observation points, the average ratio remains very high (around 90%). Note that

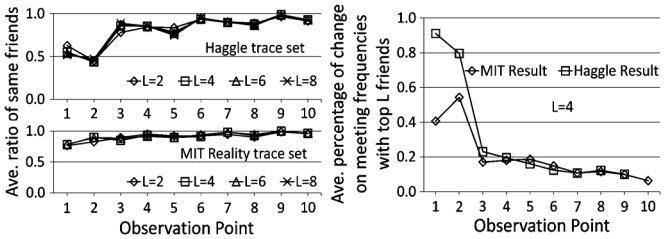



Fig. 3. Evolution on the change of friend list and meeting frequency. (a) Evolution of top $L$ friends. (b) Evolution of the change of meeting frequency with top $L$ (4) friends.

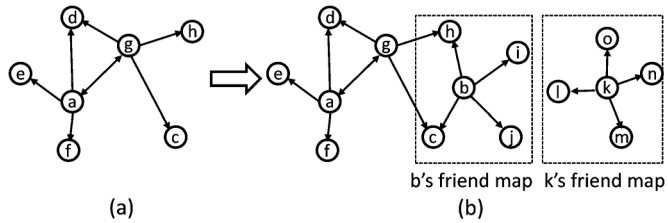



Fig. 4. Social map update process. (a) Initial social map of node $a$. (b) After node $a$ meets $b$ and $k$.

TABLE II
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 |
| ... | ... | ... |

the length between two observation points is quite long 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. We first ran the trace and selected the top $L$ most frequently met nodes as the top $L$ friends in this measurement. The frequency change of a node to its friend is measured by $|f_{i+1} - f_i|/f_i$, in which $f_{i+1}$ and $f_i$ denote the meeting frequency with the friend at observation point $i$ and $i+1$, respectively. Fig. 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.

Though the above results are obtained from only two traces with around 100 nodes, they can represent human-based DTNs to a certain extent. First, the two traces represent two typical human-based DTN scenarios (campus and conference site). Second, such results match our daily experiences that we often meet the same group of people regularly, e.g., colleagues, family members, and friends.

*b) Social Map Construction Process:* We first introduce a concept of *friendship rank*. We divide high meeting frequencies in the system to a number of ranges and assign a rank to each top $L$ friend based on meeting frequencies. Each node then matches its meeting frequencies with other nodes to these ranges to determine its friendship ranks with other nodes. The ranges can be determined using both centralized and distributed methods. In the centralized method, the system administrator predetermines the ranges based on the application scenario. To obtain the meeting frequencies among nodes in the system, a collector can be placed in a popular place to collect the meeting frequencies from nodes. In the distributed method, nodes exchange their meeting frequencies in a gossip manner. Then, when the designated node has collected the meeting frequencies of most nodes in the system, it decides the meeting frequency ranges and informs all other nodes through broadcasting. Therefore, this method has a high overhead. The system owner can select a suitable method based on the application requirement. As observed from Fig. 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. The reason we use the rank instead of meeting frequency is that it can reduce the cost in social map updates caused by the fluctuation of meeting frequencies.

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 II. A node's social map is constructed by connecting all nodes in its social table, and each node only appears once in the map. A directional link from node $i$ to node $j$ means node $j$ is in the top $L$ friend list of node $i$. Fig. 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 $L$ connected nodes in the social map accordingly. Otherwise, node $a$ integrates $b$'s friend map into its social map. Fig. 4 demonstrates the update process of node $a$'s social map after it meets node $b$ and $k$. 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 Fig. 4(b). In this case, we still regard it as part of the social map because: 1) the partition still 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 significant resources since a node's top $L$ friends usually are stable, as shown in Fig. 3(a).

The above algorithm finally generates social maps that are not identical in all nodes and may not include all 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.

*c) Social map coverage:* We define the *coverage* of the social map as the number of nodes in the map divided by the total number of nodes in the system. We then measured the average coverage of the social map on each node at 10 evenly distributed observation points with different $L$ values. The results with the Haggle trace and the MIT Reality trace are shown in

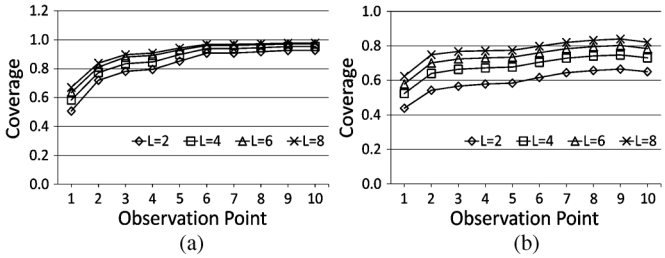



Fig. 5. Social map coverage with different $L$'s. (a) Haggle trace. (b) MIT reality trace.

Fig. 5(a) and (b), respectively. We see that even when $L = 2$, after a short period of time, the social map on each node can cover over 80% of nodes in the Haggle trace and 60% of nodes in the MIT trace. When $L = 8$, the social map coverage in the two traces increase to 90% and 80%, respectively. Such results demonstrate the feasibility of using social map to provide routing guidance.

*d) Determining the Value of $L$:* Clearly, the value of $L$ affects the social map on each node. When $L$ increases, the social map contains more information and can provide better routing guidance. However, larger $L$ also consumes more communication and storage resources. Therefore, we need to balance the value of $L$ and the cost. As aforementioned, the goal of the social map is to reflect stable social relationships. Then, we can determine $L$ based on the number of stable friends of each node. The stable friend list of a node contains the frequently met nodes whose meeting frequencies with the node are larger than half of that of the most frequently met node. For example, suppose node $j$ is the most frequently met node of node $i$, and the meeting frequency between them is $f_{ij}$. Then, nodes whose meeting frequencies with node $i$ are larger than $f_{ij}/2$ are stable friends of node $i$. The determination of $L$ can be realized in both a centralized and distributed manner.

In the centralized method, we first let the node run for a period of time so that all nodes can collect enough amount of encountering records. Then, we periodically calculate $L$ as the average number of stable friends each node has. The updated $L$ is sent to each node whenever its connection to the central server is established. In the distributed method, considering different nodes have different social relationships and, consequently, different numbers of stable friends, we do not require the $L$ to be identical among all nodes. Each node directly uses the size of its stable friend list as $L$.

Both the two methods have advantages and disadvantages. In the centralized method, with the determined $L$, mobile nodes can save the cost on exchanging $L$ value. However, it suffers from the problem that it cannot adjust $L$ timely and needs a central/super node to collect necessary information. In the distributed method, the social maps get updated timely when the stable friends of a node change, providing more accurate routing information for packets. However, the drawback of the distributed method is that it is hard to control the size of social map since individual nodes decide their own $L$s.

Some may question that the social map constructed with limited $L$ may fail to reflect some forwarding opportunities, especially for active nodes, since each node can only have most $L$ friends 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 as the top $L$ friends of more nodes, thus having more links in the social map. Therefore, the specified $L$ does not compromise routing performance, which is verified in our analysis in Section III-C.3 and experiments in Section IV.

*e) Cost Saving Resulted From Social Map:* We assume the average value of $L$ on each node is $K$ no matter if different nodes have the same $L$ or different $L$s. Then, on average, two encountered nodes only exchange their top $K$ friends and associated friendship ranks for social map construction. Suppose the size of the information of one friend is $T$ bytes, and the total amount of data exchanged is about $2TK$ bytes. In previous methods, two encountered nodes exchange their delivery abilities to the destinations of all packets to make a forwarding decision. The size of each delivery ability can be 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 $2T(N-K)M$, where $M$ is the total number of encounters. Recall that $N$ is bounded by the total number of nodes in the system. Also, the higher the packet generation rate is, the larger $N$ tends to be (suppose packet destinations are evenly distributed). On the other hand, $K$ usually is small ($K < 10$). Therefore, the social map can reduce the information exchange cost in most cases (i.e., when the packet generation rate ensures $N > K$). Furthermore, the cost saving is more valuable when the packet generation rate is high because more packets require more storage/forwarding resources in the resource-limited DTNs.

*2) Social Link Weight Calculation:* We assign a weight to the link connecting two nodes, say nodes $i$ and $j$, in a social map to represent how fast a packet can be forwarded between them. We consider two factors in this process: the meeting frequency and the social closeness between the two nodes, which are reflected by shared top $L$ friends [12], respectively. We consider social closeness for weight calculation because people with close relationship are likely to share the same group of friends [12]. 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 message delivery ability between the two connected nodes.

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

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

where $r_k$ denotes the rank of the $k$th link. For two nodes, the weight of 1-hop route reflects their meeting frequency while the 2-hop routes show the social closeness.

Then, the weight for a link in the social map connecting nodes $a$ and $b$, denoted $W_{ab}$, integrates all 1-hop and 2-hop routes between them

$$W_{ab} = \frac{1}{\sum_{s=0}^{m-1} w_{ab_s}} \tag{2}$$

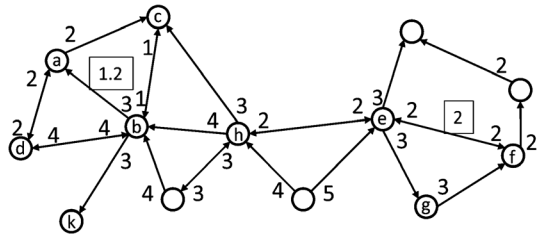



Fig. 6. Part of node $h$'s social map.

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

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

*3) Understanding the Social Link Weight:* The friendship rank can represent the expected intermeeting time between the two connected nodes, i.e., the expected delay for a packet to be relayed between the two nodes, as explained in the definition of friendship rank. As a result, the sum of each link's friendship rank in a route [i.e., $\sum_{k=0}^{l-1} r_k$ in (1)] represents the expected delay of replaying a packet through the route. Then, the weight of a route, calculated in (1), actually represents the "normalized throughput" of the route. Note in the above discussion that we say "normalized throughput" because we assume each node's memory can only hold one packet. This can be adapted to practical scenarios by deciding the average amount of memory on each node.

The overall "normalized throughput" between two nodes, say $a$ and $b$, is the sum of each route's "normalized throughput": $\sum_{s=0}^{m-1} w_{ab_s}$. Then, the weight of the social link connecting two nodes, which is calculated as 1 over the "overall throughput" [(2)], represents the expected delay to relay a packet from one node to the other node through the possible routes considered in our design.

### C. Social-Map-Based Routing Algorithm

In this section, we first introduce how we utilize the social link weight and then present the detailed routing process.

*1) Deciding Delivery Ability:* In DTN routing, a packet is always forwarded to the candidate forwarder that has the highest ability to deliver it to its destination. Then, how do we decide the delivery ability with the social link weight?

For two nodes in the social map, there are multiple paths connecting them. We define the weight of a path on the social map as the sum of the weights of its social links. Since the weight of a social link denotes the expected delay to forward a packet between the two connected nodes, the weight of a path represents the expected delay to pass a packet through the path. Then, the weight of the minimal weight path from the holder to the destination of a packet represents the minimal expected time needed to deliver the packet. Since delay is the crucial factor in DTN routing, we take the weight of the minimal weight path to represent the delivery ability between the two nodes. This also matches the general DTN routing process, which forwards packets hop by hop and aims to maximally reduce the expected delay in each forwarding hop.

*2) Routing Process:* With above analysis, the guideline of our routing algorithm is to always forward a packet to a node whose minimal-weight path to the destination node has smaller weight, i.e., gradually and maximally reducing the minimal expected delay. In detail, suppose node $a$ needs to decide whether node $b$ is a better forwarder for one of its packets. Node $a$ first checks whether both $a$ and $b$ are disconnected to the destination node in the social map. If yes, we use a backup metric to decide the forwarder, as introduced later. Otherwise, node $a$ uses the Dijkstra [21] algorithm to find the minimal-weight paths from the destination node to $a$ and $b$. Note if $a$ or $b$ is disconnected to the destination node, the weight of its minimal-weight path to the destination node is the maximal value. If node $b$'s minimal-weight path has smaller weight than that of node $a$, node $a$ would forward the packet to node $b$.

Moreover, 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 summarize the routing algorithm.

*a) Incomplete Social Map:* As stated previously, the social map on each node may only cover part of the entire network. Though we find in Section III-B.1.c that the social map has high coverage, it is possible that when a node meets another node, the destination node of a packet is disconnected from the two nodes in the social map (i.e., no path can be found for both nodes). 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 associated with the node in the social map. The more links connecting to a node, the more active it is. Then, the packet is forwarded to the node with higher active degree. 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 [15].

*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 to record 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 pseudocode shown in Algorithm 1.

1) When two nodes meet with each other, they first exchange their friend maps, which are then used to update their social maps (lines 2–4). After this, each node processes its packets sequentially (line 7).
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 9).

**Algorithm 1:** Pseudocode of the SMART routing algorithm executed by $a$ upon meeting node $b$.

```
1:  procedure EXCHANGETOPFRIENDSWITH(b)
2:      n.sendTopFriendsTo(b)
3:      n.receiveTopFriendsFrom(b)
4:      n.updateSocialMap()
5:  end procedure
6:  procedure SELECTFORWARDER(b)
7:      for each packet p in node a do
8:          bForward ← false
9:          if p.hasBeenOn(b) = false then
10:             if a.connect(p.des) || b.connect(p.des) then
11:                 if b.getW(p.des) < a.getW(p.des) then
12:                     bForward ← true
13:                 end if
14:             else
15:                 if b.getDegree( ) > a.getDegree( ) then
16:                     bForward ← true
17:                 end if
18:             end if
19:             if bForward = true then
20:                 a.forwardPacketTo(p, b)
21:             end if
22:         end if
23:     end for
24: end procedure
25: procedure RECEIVEPACKETSFROM (p, b)
26:     while Memory.Full() = true do
27:         dropOldestPacket()
28:     end while
29:     if p.NotDropped() = true then
30:         p.InsertID(a)
31:         StorePacket(p)
32:     end if
33: end procedure
```

3) If the destination node connects to at least one of the two nodes in the social map, the node checks whether the other node's minimal-weight path to the destination node has lower weight with the Dijkstra algorithm. Otherwise, the node checks whether the other node has higher active degree. The processing proceeds to step 4 if yes to either of above check, and step 5 otherwise (lines 10–21).
4) The node forwards the packet to the other node. When the other node receives the packet, if the storage is full, the oldest packet is dropped until the memory is not full. Then, if the new packet is not dropped, the node inserts its ID into the packet and stores the packet (lines 25–33).
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 7).

In summary, two encountering nodes in SMART only exchange a small amount of information for social map construction and forwarder selection. Also, the delivery ability in SMART naturally considers the multihop relay through the top $L$ friends. This global view-based forwarder selection can enable more efficient routing. SMART can also support different routing metrics such as minimal average/maximal delay and minimal missed deadlines by setting different packet forwarding priorities when two nodes meet [8]. SMART uses first-come–first-out forwarding sequence in this paper.

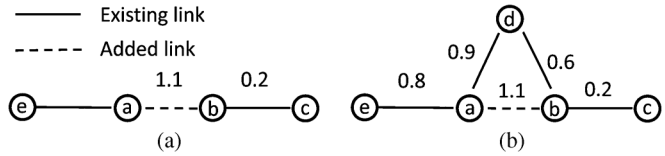



Fig. 7. Improvement on social map when $L$ increases. (a) Connecting disconnected nodes. (b) Improve existing path ($e$ to $c$).

*3) Effect of Top L Friends:* Recall that SMART only allows each node to keep a node's top $L$ friends in the social map. We now analyze how $L$ affects the routing efficiency and further check the correctness of our design (i.e., only store top $L$ friends) based on the routing procedure.

As aforementioned, the key step in the routing procedure is to find the minimal weight path from the candidate forwarders to the destination in the social map. The precision of discovered minimal weight path decides the effectiveness of the forwarder selection and consequently, the routing efficiency. We discuss the scenario when we gradually increase $L$ from 1. When $L$ increases, each node can initiate more links in the social map. The added links can help find a better minimal weight path (i.e., with a smaller weight) by two ways.

- It connects two previously disconnected nodes and enables a new minimal weight path, as shown in Fig. 7(a).
- It improves an existing minimal weight path by lowering its weight, as shown in Fig. 7(b).

Clearly, when $L$ is small, the two cases happen easily if $L$ increases, thereby improving the accuracy of discovered minimal weight path. However, when $L$ is large, we argue that both cases are not easy to happen.

First, when $L$ is large enough, the graph is almost connected. The first case then can hardly happen. Second, for the second case, by examining Fig. 7(b), we find that it happens when the weight of the added link (1.1) is smaller than the sum of the weights of the two previous links connecting the two nodes ($0.9 + 0.6$). Then, since the links are added incrementally, i.e., the newly added link for a node is its $(L + 1)$th friend when $L$ increases to $L+1$; when $L$ is large, the weights of the new links are large, thereby it can hardly satisfy the requirement for the second case.

In conclusion, when $L$ increases from a small value, the two cases happen easily and improve the routing efficiency. However, when $L$ is larger enough, increasing it can hardly trigger the two cases to improve the routing efficiency. Such a result demonstrates the correctness of only storing top $L$ friends (with a proper $L$). Such a finding matches our discussion in Section III-B.1 that people forward messages mainly through major social relationships.

*4) Advanced Extension of Packet Routing:* We further present two extensions of SMART that improve its efficiency.

*a) Alleviating Load on Overloaded Nodes:* In the routing protocol, active nodes that are a top $L$ friend of more nodes have more chances to be selected as the packet forwarder, and hence can easily become overloaded. To avoid overloading such nodes, we introduce an overload bit in the beacon messages. When a node is about to be overloaded, it sets the overload bit

TABLE III
BFT IN NODE $a$

| Destination | Better forwarder (path weight) |
|---|---|
| $b$ | $k(1.8), e(1.7), h(0.4), o(0.9)$ |
| $c$ | $b(2.2), c(1.2), j(1.4), f(0.8)$ |
| $f$ | $k(0.8), m(0.9), d(1.3)$ |
| ... | ... |

in its beacon messages. Then, its neighbors will not select it as a packet forwarder candidate. The overload bit is reset when the load on the node recovers to a normal level.

*b) Reducing Computation Cost:* Recall that when $a$ needs to decide whether the newly met node $b$ is a better forwarder for one of its packet, the Dijkstra algorithm regards the destination as the root node and iteratively selects the node that has the minimal path weight to the destination. The selection process ends when either $a$ or $b$ is picked up, and the first picked node in $a$ and $b$ is a better carrier for the packet. The complexity of this process is $O(n^2)$ [22], where $n$ is the number of nodes in the social map. This process is executed for the destination of each packet in node $a$. Therefore, the computation load is very high without further optimization.

In order to reduce the computation load, we propose to proactively cache better forwarders for each destination node. Specifically, for each destination, node $a$ runs the Dijkstra algorithm until $a$ is picked or all nodes are picked. Then, the nodes that are selected prior to $a$ and their path weights to the destination are stored in a Better Forwarder Table (BFT). Thus, the BFT records the nodes that have higher delivery ability to the destination node than node $a$. Table III shows an example of a BFT in node $a$. Later on, when node $a$ encounters these recorded nodes, it forwards the packets for corresponding destination to them directly without running the Dijkstra algorithm. If multiple nodes in BFT for the same destination are met at the same time, the one with the minimal path weight is selected as the forwarder.

The BFT should be updated timely to reflect the changes on the social map due to meeting frequency change in DTNs. It is updated when either the destination of a packet is absent from the table or the number of changed social links constitutes more than $TH_c\%$ of all social links in the social map. Note each node records the number of added or removed social links in its social map to track the second case. When the former happens, the Dijkstra algorithm is launched to make the forwarding decision for the destination and meanwhile update the corresponded row in the BFT. When the latter happens, all entries in the BFT become invalid and are removed, after which the BFT is updated following the first scenario.

We then discuss the maintenance and storage cost of the BFT. First, the BFT table updates frequently at the beginning when the social map has less information. However, as mentioned in Section III-B.1, a node's most frequently met nodes are very stable. Thus, after the initial stage, the social map on a node would become stable. Then, we can set a relatively large $TH_c$, which would lead to a low update overhead for BFT. The BFT in a node only stores the IDs of better forwarders and their associated path weights for each destination. Since a pair of ID and weight only occupies several bytes, the size of a BFT would not be a burden to modern mobile devices which usually have gigabyte-level memory (i.e., 8 GB).

**Algorithm 2:** Pseudocode of the SMART BFT-based routing algorithm executed by $a$ upon meeting node $b$.

```
1: procedure SELECTFORWARDER(b)
2:     for each packet p in node a do
3:         d ← destination of packet p
4:         if both a and b are disconnected from d then
5:             Forwarder(p) ← the node in {a, b} with higher
               active degree
6:         else
7:             if there is an entry for d in BFT then
8:                 if b is in the entry then
9:                     Forwarder(p) ← b
10:                end if
11:            else
12:                LaunchDijkstraAlgFor(p)
13:            end if
14:        end if
15:    end for
16: end procedure
```

Algorithm 2 shows the process for node $a$ to decide the forwarder for its packets when it meets node $b$. For each of its packets, node $a$ first checks if both $a$ and $b$ are disconnected to the packet's destination in the social map. If so, the forwarder should be the one with the higher active degree. Otherwise, node $a$ checks whether the entry for the destination node exists in its BFT and node $b$ exists in the entry. If so, it forwards the packet to node $b$ directly. If not, node $a$ uses the Dijkstra algorithm to decide the forwarder as previously described in Section III-C.2 and meanwhile updates its BFT.

### D. Discussion on Scalability and Security

We further briefly discuss the scalability and security issues of SMART.

*1) Scalability of SMART:* Although each node in SMART needs to store many friend maps, SMART is scalable on storage in a large-scale network with 10 000 nodes due to two reasons. First, one friend map only contains $L$ IDs and $L$ friendship ranks, which occupy about $8L$ bytes. Then, 10 000 friend maps require about $80L$ kB memory, which is not a big burden for most mobile devices nowadays. Second, a node only stores the friend maps of nodes it has met, which are very limited in a large network since it mostly only meets nodes in the local community. Therefore, even when the network size is very large, the storage consumption in SMART is limited. As mentioned in Section I, SMART is designed mainly for applications in a certain local community, which means the network size usually is not very big. In our experiment, both traces contain less than 100 nodes. Then, if $L = 8$, the social map on a node needs at most 6.4 kB. Thus, the memory can be satisfied easily in potential scenarios.

With BFT, the Dijkstra algorithm runs only when necessary. The complexity to determine packet forwarder by checking the better forwarder table is $O(n)$. Recall that without BFT, the complexity to determine packet forwarder using the Dijkstra algorithm is $O(n^2)$. Therefore, the computation complexity is greatly reduced with BFTs. Our experimental results in

Section IV-D show that the BFT greatly reduces the computation cost without greatly compromising the routing efficiency.

SMART is also scalable regarding routing path lengths. Large social networks have a 6-hop property, in which two unknown persons can be connected by 6 hops of transits on average [23]. This implies that through top $L$ friends, SMART may not need a large number of hops to deliver a packet to its destination even when the network is very large. Note that SMART is designed for social networks in local communities where the connections between people usually are tight. Thus, in SMART, a node needs fewer hops to reach an unknown person. Our experimental results also show that the average number of forwarding hops for a successfully delivered packet is about 5 in tests with both real traces.

*2) Security in SMART:* In DTNs, nodes may belong to different entities or organizations. Therefore, some nodes may not follow the SMART routing algorithm or even behave maliciously for individual interests. In this paper, we briefly discuss the blackhole attack in SMART and leave other security problems to our future work.

In a blackhole attack, malicious node $a$ claims that node $b$ is its friend and falsely reports a very high friendship rank with $b$ in order to attract packets destined to $b$ and drop these packets. We propose to use friendship verification to detect and prevent it. Specifically, when node $a$ wants to take node $b$ as its top $L$ friend, node $a$ must send the friendship rank to node $b$ asking for an approval. Node $b$ then verifies the correctness of the friendship rank and signs the friendship rank with its private key if it is correct. The friendship rank can be updated and signed timely since they meet frequently (i.e., $b$ is a top $L$ friend of $a$). Later, when $a$ claims $b$ as its top $L$ friends, other nodes can verify the friendship rank with the public key of node $b$. As a result, nodes that have forged the friendship rank can be identified, thereby preventing the blackhole attack.

An advantage of the above method is that a node would not collude with another node to fake the friendship ranks since this would reduce its opportunity to receive packets destined for it. However, a node may attract and drop packets by its real friendship ranks. The detection of such behaviors is nontrivial in DTNs and is out of the scope of the paper.

## IV. Trace-Driven Performance Evaluation

We conducted event-driven experiments using real traces from the MIT Reality project [16] and the Haggle project [17]. In this paper, we first focus on the test with fixed $L$ to show how different $L$ affects the routing performance. Then, we examine the performance of SMART under different $L$'s. We also examined the performances of the method to decide $L$ distributively and the BFT. We compared SMART with the following representative DTN routing algorithms.

1) *PROPHET*: PROPHET is a probabilistic routing algorithm. It calculates delivery ability based on past encountering records and forwards packets to nodes with higher delivery ability 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 with the destination node (the number of shared encountered nodes). 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 300 s and per 40 s in the MIT trace and the Haggle trace, respectively. In the test, at most $10 * T_l$ packets can be exchanged when two nodes meet, where $T_l$ is the length of the encountering session in seconds. 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 ability 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 5000 to 25 000 (i.e., $R_n$ was varied from 1 to 5). The memory size on each node was set to 100 kB. In the test with different memory sizes, the memory on each node was varied from 60 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 destinations in the experiment.
- *Normalized average delay*: the average delay of all packets. We regard the delay of an unsuccessful packet as the trace length, which is 340 k and 2560 k s for the Haggle trace and MIT Reality trace, respectively.
- *Normalized forwarding hops*: the total number of packet forwarding hops divided by the number of successfully delivered packets.
- *Routing cost*: the number of information units exchanged between encountered nodes in the experiment.

### A. Performance With Different Packet Generating Rates

*1) Hit Rate:* Figs. 8(a) and 9(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 $\mathrm{SMART} > \mathrm{SimBet} > \mathrm{PROPHET} > \mathrm{StaticWait}$.

StaticWait shows the lowest hit rate because packets only statically wait in their generators to reach destinations. SMART deduces nodes' delivery abilities to destinations based on the relatively stable social map. It can choose an optimal forwarder in a long path to the destination with a broad view, thereby generating the highest hit rate. PROPHET and SimBet can only evaluate the delivery ability within 1 or 2 hops, which cannot consider long routing paths, leading to a lower hit rate than SMART.

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,

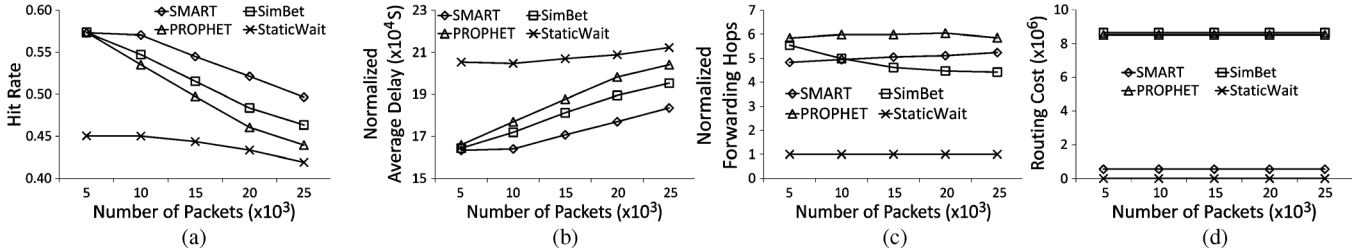



Fig. 8. Performance of each method with the Haggle trace under different packet rates. (a) Hit rate. (b) Normalized average delay. (c) Normalized forwarding hops. (d) Routing cost.

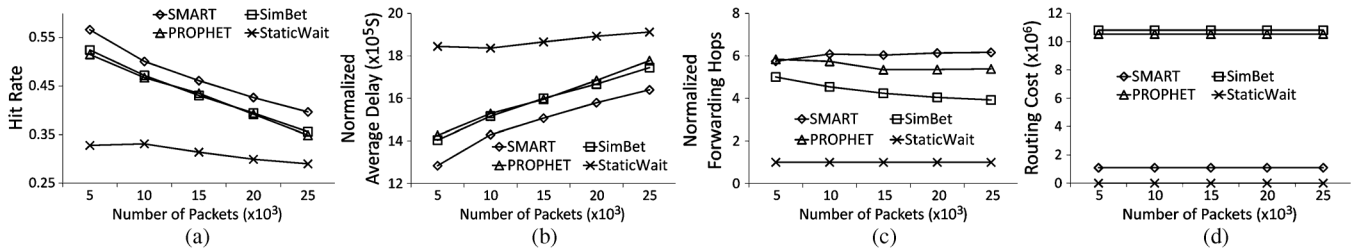



Fig. 9. Performance of each method with the MIT Reality trace under different packet rates. (a) Hit rate. (b) Normalized average delay. (c) Normalized forwarding hops. (d) Routing cost.

more packets are dropped by 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. This is because we configure fixed memory in the test. Then, when the number of packets increases, more packets are converged to certain important nodes, which have fixed storage, resulting in more dropped packets. We can adopt load balance techniques in Section III-C.4 to alleviate this problem. In StaticWait, packets are evenly distributed among nodes, thereby better utilizing the memory on all nodes and leading to fewer dropped packets.

With the 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) Normalized Average Delay:* Figs. 8(b) and 9(b) show the normalized 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 normalized average delays of the four methods follow $\mathrm{SMART} < \mathrm{SimBet} < \mathrm{PROPHET} < \mathrm{StaticWait}$. SMART has the lowest normalized average delay because it considers multihop 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. 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 normalized average delays of the four methods increase as the packet generating rate increases. This is because we regard the delay of dropped packets as the length of the test trace, which is much longer than the delay of a successful packet. Then, when the packet generating rate increases, there are more dropped packets (the hit rate decreases), as shown in Figs. 8(a) and 9(a), leading to increased normalized average delay.

*3) Normalized Forwarding Hops:* Figs. 8(c) and 9(c) show the normalized forwarding hops of the four methods with the Haggle trace and the MIT Reality trace, respectively. We observe that StaticWait has very low forwarding hops and the other three methods have high forwarding hops. In StaticWait, each packet waits in its initiator for the destination. Therefore, each successful packet is forwarded only once. The active forwarding in SMART, SimBet, and PROPHET leads to much more forwarding for each successful packet. We see that the normalized forwarding of SMART is on the same level with SimBet and PROPHET, but SMART has higher hit rate and lower average delay. This further demonstrates that the active forwarding in SMART is more effective.

*4) Routing Cost:* Figs. 8(d) and 9(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:* Figs. 10(a) and 11(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 Figs. 8(a)

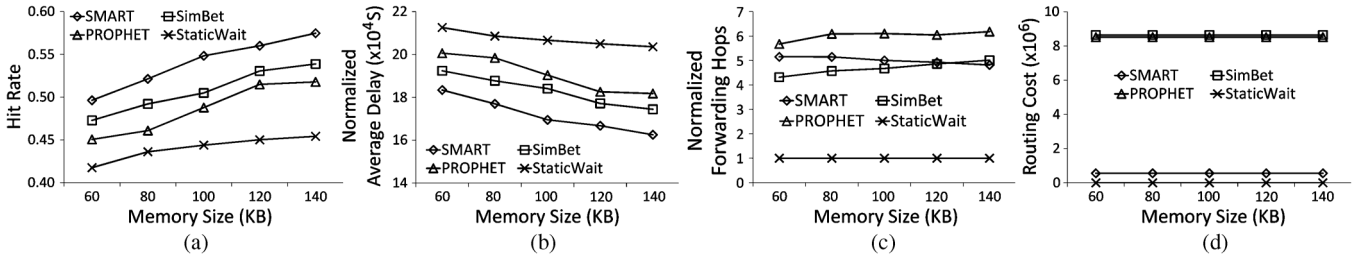



Fig. 10. Performance of each method with the Haggle trace under different memory sizes. (a) Hit rate. (b) Normalized average delay. (c) Normalized forwarding hops. (d) Routing cost.

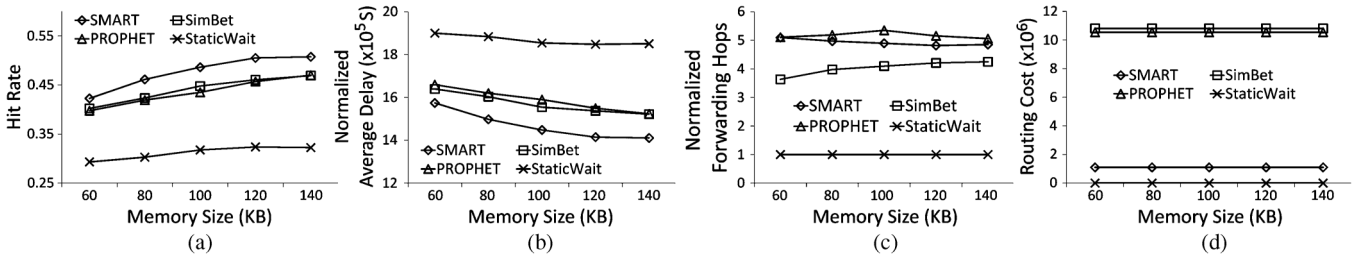



Fig. 11. Performance of each method with the MIT Reality trace under different memory sizes. (a) Hit rate. (b) Normalized average delay. (c) Normalized forwarding hops. (d) Routing cost.

and 9(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 larger memory size, each node can carry and forward more packets to their destinations, leading to a higher hit rate.

*2) Normalized Average Delay:* Figs. 10(b) and 11(b) 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 relationship of the four methods on average delay is the same as in Figs. 8(b) and 9(b) for the same reasons. It is interesting to see that when the memory size on a node increases, the normalized average delays decreases. This is because when the memory size increases, there are fewer dropped packets, whose delay is very large (i.e., trace length), leading to decreased normalized average delay.

*3) Normalized Forwarding Hops:* Figs. 10(c) and 11(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 normalized forwarding hops follow the same trend as in Figs. 8(c) and 9(c) due to the same reasons.

*4) Routing Cost:* Figs. 10(d) and 11(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 in Figs. 8(d) and 9(d) for the same reasons. This is because the routing cost is irrelevant to the number of packets on each node, but is 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.

### C. Effect of the Value of L

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 Tables IV and V.

TABLE IV
ROUTING PERFORMANCE WITH THE HAGGLE TRACE

| $L$ | Hit Rate | N. Ave. Delay (s) | N. Fwd. Hops | Routing Cost |
|---|---|---|---|---|
| 2 | 0.541425 | 172,614.8 | 4.992 | 277,704 |
| 3 | 0.551023 | 169,006.2 | 4.932 | 416,556 |
| 4 | 0.551156 | 168,715.7 | 5.050 | 555,408 |
| 5 | 0.560021 | 165,903.7 | 4.945 | 833,112 |
| 6 | 0.56822 | 163,745.5 | 4.860 | 833,112 |
| 7 | 0.572219 | 162,426.5 | 4.825 | 971,964 |
| 8 | 0.576218 | 161,251.9 | 4.775 | 1,110,816 |

TABLE V
ROUTING PERFORMANCE WITH THE MIT REALITY TRACE

| $L$ | Hit Rate | N. Ave. Delay (s) | N. Fwd. Hops | Routing Cost |
|---|---|---|---|---|
| 2 | 0.431467 | 1,575,819.5 | 4.028 | 546,892 |
| 3 | 0.459933 | 1,509,557.2 | 4.776 | 820,338 |
| 4 | 0.481467 | 1,459,355.8 | 4.858 | 1,093,784 |
| 5 | 0.502867 | 1,413,748.9 | 4.884 | 1,367,230 |
| 6 | 0.502533 | 1,410,643.2 | 5.027 | 1,640,676 |
| 7 | 0.515667 | 1,384,143.6 | 4.994 | 1,914,122 |
| 8 | 0.515933 | 1,385,610.1 | 4.955 | 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 ability 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 increase 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; thus, 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) Normalized Average Delay:* We find that the normalized 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). When the $L$ increases from a small value, the social map constructed on each node becomes more complete, resulting in better forwarder selection and decreased average delay. The increased hit rate also leads to fewer dropped packets, which have high delay (i.e., trace length). Consequently, the normalized average delay decreases. After $L$ reaches the medium value, which enables social maps to show almost all most frequently met nodes, the enhancement in the routing efficiency is not significant if $L$ further increases, as shown in the first column of the two tables. Therefore, the normalized average delays remain on the same level.

*3) Normalized Forwarding Hops:* We see that when $L$ increases from 2 to 8, the normalized forwarding hops of SMART remains stable when the Haggle trace is used and increases in the test with the MIT Reality trace. This is caused by the different environments of the two traces. In the Haggle project, nodes are more crowded, so it is easy to find a next-hop node even when $L$ is small. Therefore, the normalized forwarding hops remain stable with different $L$. However, in the MIT Reality trace, nodes are sparser, so a small $L$ cannot reflect the social structure well, leading to fewer forwarding opportunities and a low normalized forwarding hops.

*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 Figs. 8(d) and 9(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:* The 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 Figs. 8 and 9. Also, the performance of SMART is improved when $L$ increases and remains stable when $L$ is larger than 5. Such a result matches our analysis in Section III-C.3 and justifies the idea that top $L$ friends can provide enough 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.

### D. Effect of Dynamic $L$ and BFT-Based Routing

In Section III-B.1.d, we proposed two methods to determine $L$: centralized method and distributed method. In the former, $L$ is predetermined based on network statistics, and all nodes have the same $L$. In the latter, $L$ is dynamically decided on each node. In Section IV-C, we already presented the performance of SMART with different $L$ values in the centralized method. In this section, we present the performance of SMART with dynamically determined $L$ values in the distributed method, denoted by SMART-DL. In SMART-DL, $L$ is determined so that the $L$th most frequently met node has half of the meeting frequency of the most frequently met node. We also present the performance of the BFT-based routing introduced in Section III-C.4.b, denoted by SMART-BFT. In the test, we set the memory on each node to a medium value of 100 kB, $L$ to a medium value of 4 in SMART and SMART-BFT, and the threshold of the percentage of changed links in a social map for BFT update in SMART-BFT to 20%.

*1) Hit Rate:* Figs. 12(a) and 13(a) show the hit rates of SMART, SMART-DL, and SMART-BFT with the Haggle trace and the MIT Reality trace, respectively. We see that in both traces, SMART-DL achieves higher hit rate than SMART. This is because in SMART-DL, dynamically determined $L$'s can reflect the major social relationships with all nodes in the social map, leading to a higher hit rate. We found that the average value of $L$ equals around 5–6 in SMART-DL, which is larger than $L = 4$ in SMART. We also see that SMART-BFT has a slightly lower hit rate than SMART-BFT. The BFT in SMART-BFT is updated only when the percentage of the changed links in a social map is larger than a threshold (i.e., 20%), leading to insufficiently accurate routing guidance for some packets and hence slightly degraded hit rate. However, the hit rate decrease is very minor while SMART-BFT significantly reduces the computation cost in forwarder determination in the routing process, as shown below.

*2) Normalized Average Delay:* Figs. 12(b) and 13(b) show the normalized average delays of SMART, SMART-DL, and SMART-BFT with the Haggle trace and the MIT Reality trace, respectively. We find that SMART has larger average delay than SMART-DL in the tests with both the traces. This is because the social maps in SMART-DL can reflect more social relationships. Therefore, it can guide the packet forwarding more accurately, leading to lower average delay. We also see that SMART-BFT has slightly larger average delay than SMART. For SMART-BFT, it cannot reflect the changes on social maps timely, which lowers the accuracy of forwarder selection, resulting in higher average delay than SMART.

*3) Normalized Forwarding Hops:* Figs. 12(c) and 13(c) show normalized forwarding hops of SMART, SMART-DL, and SMART-BFT with the Haggle trace and the MIT Reality trace, respectively. We see that in both traces, the normalized forwarding hops follow $\mathrm{SMART-DL} > \mathrm{SMART} > \mathrm{SMART-BFT}$. With the dynamic $L$ value determination, the social maps in SMART-DL can reflect more close social relationships, leading to more packet forwarding opportunities and consequently more packet

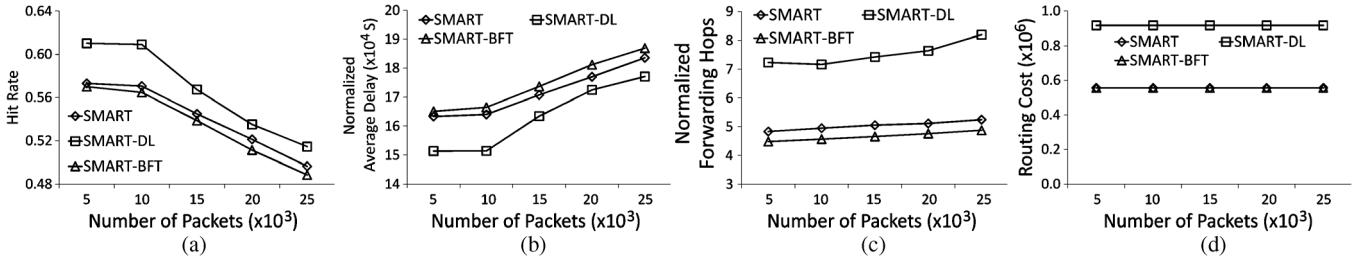



Fig. 12. Performance of each extension with the Haggle trace under different packet rates. (a) Hit rate. (b) Normalized average delay. (c) Normalized forwarding hops. (d) Routing cost.

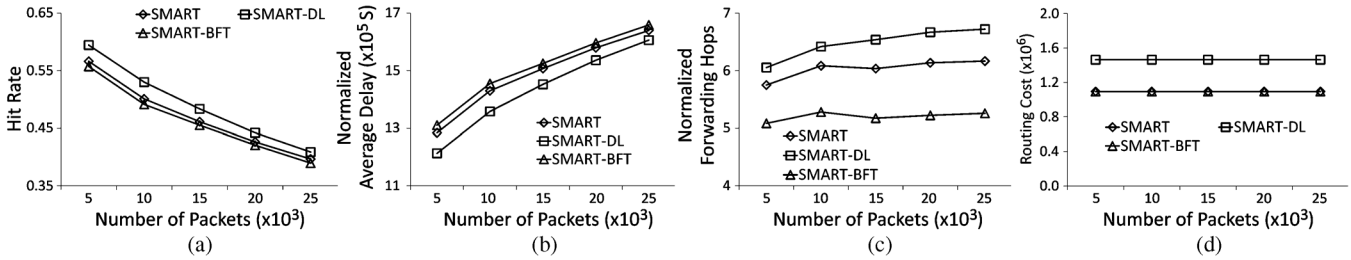



Fig. 13. Performance of each extension with the MIT Reality trace under different packet rates. (a) Hit rate. (b) Normalized average delay. (c) Normalized forwarding hops. (d) Routing cost.

forwarding. On the contrary, the untimely updated better forwarder table in SMART-BFT reduces the number of packet forwarding opportunities, leading to less packet forwarding than SMART. These results match those in Figs. 12(a) and 13(a) since higher hit rate (i.e., successful deliveries) comes at more forwarding hops.

*4) Routing Cost:* Figs. 12(d) and 13(d) show the routing costs of SMART, SMART-DL, and SMART-BFT with the Haggle trace and the MIT Reality trace, respectively. We find that SMART-BFT and SMART generate the same routing cost and SMART-DL produces higher routing cost. Both SMART-BFT and SMART have fixed $L = 4$, leading to the same routing cost. Nodes in SMART-DL dynamically adjust $L$. We found that the average values of $L$ in SMART-DL are around 5–6, leading to more information exchange among top $L$ friends.

Fig. 14(a) shows the average values of $L$ at 10 observation points, which are evenly distributed in each trace. We find that after half of each trace, the average values of $L$ are about 6 and 5 in the Haggle and MIT Reality traces, respectively. This result shows the average number of stable friends of each node in the two traces. Comparing the results in Figs. 12(a) and 13(a) and those in Tables IV and V, we further find that SMART-DL achieves higher hit rate than SMART with fixed $L = 6$ and $L = 5$. This result demonstrates that the proposed method to determine $L$ dynamically can effectively find the important social relationships in the network.

*5) Computation Cost Reduction in SMART-BFT:* We further evaluate the effect of SMART-BFT on reducing the computation cost with different packet generating rate. Since the Dijkstra algorithm used in the routing has the most computation cost, we measure the computation cost as the number of Dijkstra algorithm launched in the experiment. The test results are shown in Fig. 14(b), in which the name after a method denotes the trace used in the experiment. We find that the BFT can save up to 65% of computation cost in the tests with both traces. Combining the results in Figs. 12 and 13, we conclude

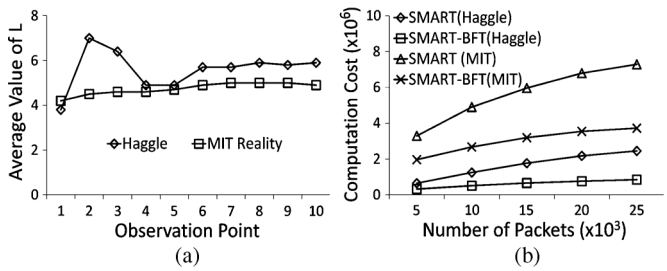



Fig. 14. Experiment results on average value of $L$ and computation cost. (a) Average value of $L$. (b) Computation cost.

TABLE VI
EFFICIENCY AND COST IN THE GENI TEST

| Method | Hit Rate | N. Ave. Delay (s) | N. Fwd. Hops | Routing Cost |
|---|---|---|---|---|
| SMART | 0.511 | 1,380,588.7 | 5.47 | 1,103,472 |
| SimBet | 0.496 | 1,423,667.5 | 4.35 | 14,173,600 |
| PROPHET | 0.501 | 1,413,484.8 | 3.24 | 12,315,306 |
| StaticWait | 0.31 | 1,861,894.3 | 1.04 | 0 |

that the BFT-based routing can significantly reduce the computation cost of SMART without greatly compromising the routing efficiency.

## V. GENI EXPERIMENT

We further evaluated the performance of the four methods on the real-world GENI ORBIT testbed [24], [25]. 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.

The test results are shown in Table VI. We see that SMART produces the highest hit rate, the lowest normalized average delay, and the second lowest cost compared to other 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, which is a lightweight routing algorithm in delay-tolerant networks that utilizes distributed social maps on mobile nodes. By exploiting the social network property that a person's most frequently encountered friends often remain stable, SMART enables each node to build a social map to record 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 ability between two nodes is evaluated by considering both meeting frequency and social closeness. Then, packets are forwarded to nodes with higher delivery ability 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 of SMART in comparison to previous algorithms.

Currently, the routing performance of SMART is limited by the intermittent connections in DTNs. Therefore, SMART is only suitable for delay-tolerant applications such as distributed friend or community recommendation based on encountering records, distributed Question&Answer systems, and the sharing of files that are tolerant to delay. In the future, we plan to further improve the routing performance of SMART for delay-sensitive applications and explore other interesting and novel applications in human-based DTNs.

## References


[1] S. Jain, K. R. Fall, and R. K. Patra, "Routing in a delay tolerant network," in *Proc. SIGCOMM*, 2004, pp. 145–158.

[2] A. Pentland, R. Fletcher, and A. Hasson, "Daknet: Rethinking connectivity in developing nations," *Computer*, vol. 37, no. 1, pp. 78–83, Jan. 2004.

[3] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Pocket switched networks and human mobility in conference environments," in *Proc. WDTN*, 2005, pp. 244–251.

[4] M. Motani, V. Srinivasan, and P. Nuggehalli, "PeopleNet: Engineering a wireless virtual social network," in *Proc. MobiCom*, 2005, pp. 243–257.

[5] A. Vahdat and D. Becker, "Epidemic routing for partially-connected *ad hoc* networks," Duke University, Durham, NC, USA, Tech. Rep., 2000.

[6] A. Lindgren, A. Doria, and O. Scheln, "Probabilistic routing in intermittently connected networks," *Mobile Comput. Commun, Rev.*, vol. 7, no. 3, pp. 19–20, 2003.

[7] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "MaxProp: Routing for vehicle-based disruption-tolerant networks," in *Proc. IEEE INFOCOM*, 2006, pp. 1–11.

[8] A. Balasubramanian, B. N. Levine, and A. Venkataramani, "DTN routing as a resource allocation problem," in *Proc. SIGCOMM*, 2007, pp. 373–384.

[9] 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. IEEE INFOCOM*, 2010, pp. 1–9.

[10] F. Li and J. Wu, "MOPS: Providing content-based service in disruption-tolerant networks," in *Proc. IEEE ICDCS*, 2009, pp. 526–533.

[11] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: Social-based forwarding in delay tolerant networks," in *Proc. MobiHoc*, 2008, pp. 241–250.

[12] E. M. Daly and M. Haahr, "Social network analysis for routing in disconnected delay-tolerant MANETs," in *Proc. MobiHoc*, 2007, pp. 32–40.

[13] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco, "Socially-aware routing for publish-subscribe in delay-tolerant mobile *ad hoc* networks," *IEEE J. Sel. Areas Commun.*, vol. 26, no. 5, pp. 748–760, Jun. 2008.

[14] C. Boldrini, M. Conti, J. Jacopini, and A. Passarella, "HIBOp: A history based routing protocol for opportunistic networks," in *Proc. IEEE WoWMoM*, 2007, pp. 1–12.

[15] 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. IEEE INFOCOM*, 2006, pp. 1–13.

[16] N. Eagle, A. Pentland, and D. Lazer, "Inferring social network structure using mobile phone data," *PNAS*, vol. 106, no. 36, pp. 15274–15278, 2009.

[17] A. Chaintreau, P. Hui, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Impact of human mobility on the design of opportunistic forwarding algorithms," in *Proc. IEEE INFOCOM*, 2006, pp. 1–13.

[18] A. Nelson, J. R. Batts, Jr., and R. L. Beadles, "A computer program for approximating system reliability," *IEEE Trans. Rel.*, vol. R-19, no. 2, pp. 61–65, May 1970.

[19] P. M. Lin, B. J. Leon, and T. C. Huang, "A new algorithm for symbolic system reliability analysis," *IEEE Trans. Rel.*, vol. R-25, no. 1, pp. 2–15, Apr. 1976.

[20] W. P. Dotson and J. O. Goblen, "A new analysis technique for probabilistic graphs," *IEEE Trans. Circuits Syst.*, vol. 26, no. 10, pp. 855–865, Oct. 1979.

[21] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.

[22] L. Goodman, A. Lauschke, and E. W. Weisstein, "Dijkstra complexity," 2013 [Online]. Available: http://mathworld.wolfram.com/DijkstrasAlgorithm.html

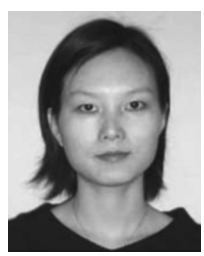
[23] S. Schnettler, "A structured overview of 50 years of small-world research," *Social Netw.*, vol. 31, pp. 165–178, 2009.

[24] "GENI project," [Online]. Available: http://www.geni.net/

[25] "ORBIT," [Online]. Available: http://www.orbit-lab.org/


**Kang Chen** (S'13) received the B.S. degree in electronics and information engineering from Huazhong University of Science and Technology, Wuhan, China, in 2005, and the M.S. degree in communication and information systems from the Graduate University of Chinese Academy of Sciences, Beijing, China, in 2008, and is currently pursuing the Ph.D. degree in electrical and computer engineering at Clemson University, Clemson, SC, USA.

His research interests include mobile ad hoc networks and delay-tolerant networks.



**Haiying Shen** (M'07–SM'13) received the B.S. degree in computer science and engineering from Tongji University, Shanghai, China, in 2000, and the M.S. and Ph.D. degrees in computer engineering from Wayne State University, Detroit, MI, USA, in 2004 and 2006, respectively.

She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, USA. Her research interests include distributed computer systems and networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing.

Dr. Shen is a Microsoft Faculty Fellow of 2010 and a member of the Association for Computing Machinery (ACM).