GreedyFlow: Distributed Greedy Packet Routing between Landmarks in DTNs

Kang Chen[a], Haiying Shen[b]

[a]*Department of Electrical and Computer Engineering, Southern Illinois University, Carbondale, IL, 62901 USA e-mail: kchen@siu.edu*
[b]*Department of Computer Science, University of Virginia, Charlottesville, VA, 22904 USA USA e-mail: hs6ms@virginia.edu*

Abstract

Delay Tolerant Networks (DTNs) have attracted much research interest recently due to its adaptability in areas without infrastructures. In such scenarios, moving data from one place (landmark) to another place (landmark) is essential for data communication between different areas. However, current DTN routing algorithms either fail to fully utilize node mobility or have additional requirements that cannot be satisfied easily in DTNs. Therefore, in this paper, we propose a distributed greedy routing algorithm, namely GreedyFlow, for efficient packet routing between landmarks in DTNs. GreedyFlow builds a local traffic map and a global landmark map on each node. The local traffic map indicates the node's knowledge about the amount of traffic (node transition) between landmarks in the area where it primarily visits. The global landmark map shows the distribution of landmarks in the system and is built offline. In packet routing, the global landmark map shows the general packet forwarding direction, while the local traffic map helps determine the next-hop landmark on the fastest path in the forwarding direction. As a result, packets are greedily forwarded toward their destination landmarks. We also propose advanced components to enhance the consistency of local traffic maps and exploit node-based forwarding, both of which help improve the packet routing efficiency. Extensive real trace driven experiments demonstrate the high efficiency of GreedyFlow.

Keywords: Delay tolerant networks, Landmark, Routing

1. Introduction

In delay tolerant networks (DTNs) [1], mobile nodes communicate with each other directly without the need of infrastructures during the encountering. Therefore, DTNs are suitable for areas where infrastructures are either unavailable or too costly. In these scenarios, it is desirable to be able to forward packets from one place (landmark) to another place (landmark), i.e., packet routing between landmarks, to support many practical applications.

For example, people living in mountainous villages may wish to communicate with each other through their computers. However, it is costly to build needed infrastructures or enable satellite connection in each village. In this case, DTN can be exploited to transfer data between these villages using mobile devices carried by people or vehicles moving in the area [2]. We can also enable the satellite connection in one village and rely on the DTN based packet routing to support delay tolerant applications such as email. Similarly, such a communication structure can be used to collect data from sensors attached to animals in mountain areas without infrastructures [3]. Even in areas with infrastructures, it can be an effective backup scheme to support the dissemination of important messages in extreme scenarios such as disaster and outage [4].

Packet routing between landmarks in DTNs can be implemented by always forwarding a packet to the node that is more likely to move to its destination landmark [5, 6, 7, 8, 9]. This indicates that nodes that can frequently visit a packet's destination landmark to deliver the packet. As a result, the mobility of nodes that rarely visit a packet's destination landmark often is not used to forward the packet. Thus, when the number of nodes that frequently visit the destination landmarks is limited, the packet routing efficiency is also limited.

To solve this problem, some researchers have proposed to forward a packet along a sequence of landmarks (called landmark path) to better utilize node mobility for packet routing between landmarks [10, 11, 12, 13]. In each hop, the packet is carried by a node to move from current landmark to the next landmark in the path. With such a design, all nodes moving between two consecutive landmarks on the landmark path can help forward the packet, even for nodes that rarely or never visit the packet's destination landmark. This means that node mobility is better utilized to forward packets.

However, the major drawback of these methods is that they require either base stations [10, 11] or the global traffic distribution [12] to calculate the optimal landmark path for each packet. Such requirements cannot be satisfied easily in real DTNs. First, due to the long delay in DTN routing, the global traffic information cannot be updated timely on each node. Second, in some DTN scenarios, such as battlefields and mountain areas, it is hard to build base stations. Such a limitation poses a significant challenge

*Corresponding author

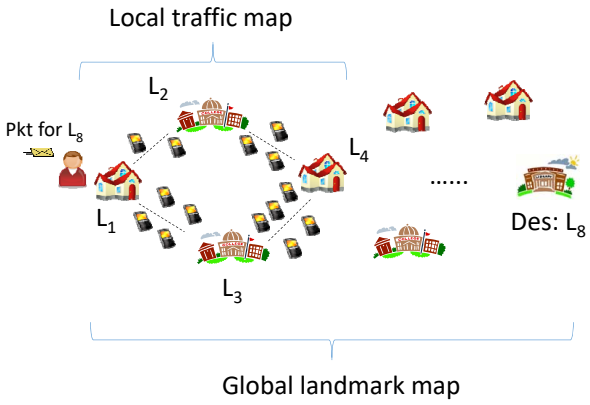


Figure 1: Illustration of the design rationale.

on realizing efficient packet routing between landmarks in DTNs.

To solve the above challenge, we propose a distributed packet routing algorithm in DTNs, denoted GreedyFlow. We assume that the network is split into sub-areas represented by landmarks. GreedyFlow follows the idea of landmark-by-landmark forwarding to better utilize node mobility. It builds a global landmark map and a local traffic map on each node (Figure 1). The global landmark map indicates the distribution of landmarks in the network and is built offline. The local traffic map reflects a node's knowledge about how frequently nodes transit between landmarks where it primarily visits. When a node meets another node, it collects the other node's transit frequencies between landmarks covered by its local traffic map to update the traffic map.

The global landmark map and local traffic map are used to guide packet routing. The basic idea is to greedily forward a packet to a landmark closest to the destination landmark within current packet holder's local traffic map (called temporary destination landmark). When a node (say N_i) receives a packet, it first determines the temporary destination landmark for the packet. Then, the node determines the fastest landmark path to the temporary destination landmark on its local traffic map and selects the next landmark on the path as the next-hop landmark. Next, N_i forwards the packet to the node that is predicted to move to the next-hop landmark.

For example, as shown in Figure 1, suppose a node in L_1 receives a packet that is destined to far-away landmark L_8. The node uses its global landmark map to identify the landmark that is closest to L_8, which is L_4. It further uses the local traffic map to find the fastest path to L_4, which is $L_1 \rightarrow L_3 \rightarrow L_4$. Then, the packet is expected to be forwarded along this path to L_4. This process repeats when the packet arrives at a new node. With such a design, the packet is always forwarded towards the landmark closest to the destination landmark through the fastest path based on local information.

The above routing paradigm is further improved from two aspects. First, we allow nodes to exchange not only their own transit frequencies but also those they have learned from others, thus improving the consistency of the distributively collected local traffic maps. Second, instead of purely relying on the landmark-by-landmark forwarding, we allow packets to be carried by nodes to reach their destination landmarks directly when the expected delay is reduced. Both the two features are designed as optional components for GreedyFlow.

GreedyFlow makes packet forwarding decisions locally without the requirement of base stations or global traffic distribution. This is the major contribution of this work over current works [10, 11, 12] that also adopt landmark-based forwarding. In summary, the contributions of this paper include

- We propose a distributed traffic map generation method that enables each node to learn the node transition frequencies between landmarks in the area where it primarily visits. We also design a feature to enhance the consistency of those traffic maps.
- We propose a fully distributed greedy algorithm that routes packets in a landmark-by-landmark manner with the local traffic map and the global landmark map, thus better taking advantage of node mobility to realize efficient packet routing between landmarks in DTNs. In addition, an advanced component is proposed to exploit node-based forwarding appropriately to further improve the routing efficiency.
- Extensive real trace based experiments demonstrate the efficiency of the proposed algorithm.

The remaining of this paper is arranged as follows. Section 2 introduces related work. Section 3 presents the detailed system design. Section 4 conducts performance evaluation through real trace driven experiments. Finally, Section 5 concludes the paper with remarks on future work.

2. Related Work

2.1. Packet Routing between Landmarks in DTNs

Packet routing between landmarks in DTNs[5, 6, 7, 8, 9, 10, 11, 12, 13, 14] has been extensively studied recently. The authors in [5] observe the long term mobility pattern of each node and use such information to forward packets to nodes that frequently move to their destinations. GeoOpps [6] routes packets to geographical locations through vehicle networks. It always forwards packets to vehicles on the route with the smallest minimal estimated time of delivery (METD). In the work of [7], a packet is always forwarded to the node that has closer distance to its destination landmark. Both the works of [8] and [9] exploit multi-copy relay to efficient forward packets to areas that may cover destination nodes. The next-hop

carrier of a packet is selected according to nodes' movement range estimated from historical location records [8] and homogeneous/heterogeneous mobility parameters [9], respectively. These methods mainly rely on nodes that are likely to visit the destination landmarks/areas. When the number of such nodes is limited, the routing efficiency is also limited.

In order to improve the efficiency of routing between landmarks, researchers have proposed to better utilize node mobility by forwarding packets in a landmark-by-landmark manner [10, 11, 12, 13, 14]. In LOUVER [10], base stations are built on road intersections for packet relay. Vehicle mobility is exploited to forward packets from one base station to another to reach the destination area. DTN-FLOW [11] expands to general DTNs. It splits the whole network into sub-areas represented by landmarks. Then, predicted node mobility is used to carry packets from one landmark to another landmark. Geomob [12] utilizes the global traffic distribution to forward packets to different areas through the landmark based relay. AAR [13] decides the weights of road segments based on the traffic distribution and uses such information to find the fastest path to reach sub-areas in vehicular delay tolerant networks. MobiT [14] takes advantage of the trajectories of different types of vehicles to forward packets to landmarks that can deliver packets to destination vehicles.

This paper follows the idea of landmark-by-landmark forwarding to improve the routing efficiency between landmarks. However, current methods need either base stations or the global traffic distribution that can not be easily satisfied in DTNs. This constraints the feasibility of these methods. This work then proposes a distributed routing protocol that makes forwarding decisions locally without the need of base stations or global traffic map, which is more suitable for DTNs.

2.2. Packet Routing between Nodes in DTNs

There are already many algorithms for packet routing between nodes in DTN [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]. PROPHET [15] updates two node's future meeting probability upon their encountering and ages it over time. It always forwards a packet to the node that has a higher probability of meeting its destination. RAPID [16] and MaxContribution [17] specify different packet forwarding and storage priorities to realize different routing performances, e.g., maximal success rate and minimal delay. The work in [18] proposes to exploit transient contact patterns to select packet forwarder. EER [19] splits the replication quote of a message between encountered nodes according to their ability to meet nodes.

Considering that mobile device owners often belong to certain social networks, social networking properties have been utilized for packet routing in DTNs [20, 21, 22, 23, 24]. MOPS groups frequently encountered nodes as communities and assigns different roles to nodes with different community visiting patterns to facilitate the publish/subscribe service in DTNs. BUBBLE [21] first forwards a packet to the community that contains its destination and then routes the packet within the community. SimBet [22] considers both a node's centrality and its similarity with the packet destination to evaluate a node's suitability to carry the packet. The works in [23] and [24] exploit fixed community and transient community structure in DTNs for efficient packet routing, respectively. CAF [25] enables individual nodes to discover community structures through a Hidden Semi-Markov Model (HSMM) over historical node contact records, which is used to improve the routing efficiency. The work in [26] evaluates nodes' closeness and centrality for packet routing in mobile opportunistic networks through predicting temporal social contact patterns.

There are also some works that utilize location/movement information for packet routing between nodes in DTNs [27, 28]. GeoDTN [27] predicts two nodes' possibility of becoming neighbors based on the similarity between their geographical movement information, which is organized as a vector. Such information is then used to guide packet routing. DTFR [28] first forwards a packet to the area that its destination node is likely to appear, which is determined by the historical movement information of the node. The packet is then spread in the area to reach the destination node.

Those algorithms target on routing packets between mobile nodes and can be indirectly applied to the routing between landmarks by regarding landmarks as static nodes in the network. However, with those algorithms, nodes that rarely visit a landmark usually are not used to carry packets for the landmark. This limits the routing efficiency. Our method thus follows the landmark-by-landmark idea to better utilize node mobility for a better routing efficiency.

3. System Design

In this section, we first introduce the network modeling and design rationale. We then present how to construct the global landmark map and the local traffic map. Finally, we introduce the detailed packet routing algorithm.

3.1. Network Modeling and Design Goal

We assume a DTN consisting of n nodes, denoted by N_i ($i \in [1, n]$). We also assume that the network is split into sub-areas, each of which is represented by a landmark. A landmark is often selected as the area where nodes gather together, such as a village in the rural area and a building on the campus. This means that the landmark is just the notation of an area and does not require any base stations to be built. We assume there are m landmarks, denoted by L_j ($j \in [1, m]$). Then, node mobility can be regarded as continuous transit between landmarks. We also assume that nodes present certain landmark visit patterns, which exists in many DTNs. For example, in DTNs consisting of mobile devices carried by people in rural areas or students

on a campus, a person or a student may mainly transit between a few landmarks (i.e., villages or buildings).

The goal of this paper is to realize efficient and distributed packet routing between identified landmarks in DTNs. Such a function can support many interesting services or applications, such as data communication between rural villages, where infrastructures are too costly to build.

3.2. Rationale of System Design

In this work, we relay packets in a landmark-by-landmark manner to reach their destination landmarks. Such a strategy can better utilize node mobility for routing packet to landmarks. For example, suppose we need to forward packets from L_1 to L_{16}. With the landmark based routing strategy, these packets are forwarded through a landmark path, say $L_1 \rightarrow L_6 \rightarrow L_{12} \rightarrow L_{16}$. As a result, nodes moving between any two neighboring landmarks on the path, e.g., L_1 and L_6, can forward these packets one step closer to destination L_{16} even though these nodes rarely or never visit L_{16}. This means that more node mobility is utilized for packet routing between landmarks, leading to better routing efficiency.

3.2.1. Challenges

The key problem is how to select a suitable landmark path for each packet. It is not hard to see that the more frequently nodes move from one landmark to a neighbor landmark, say L_1 to L_6, the more quickly a packet can be forwarded from L_1 to L_6, and the smaller the expected delay of this forwarding step. Then, the expected delay of a landmark path can be calculated as the sum of the expected delays on each hop. However, how to find the landmark path with the smallest expected delay efficiently and accurately is non-trivial. This is because nodes often are sparsely distributed in DTNs. Previous methods [10, 11, 12] realize this step by either building extra base stations on each landmark [10, 11] or requiring that each node knows the global traffic distribution [12]. Unfortunately, both requirements cannot be satisfied easily in real DTNs.

3.2.2. Our Solution

GreedyFlow routes packets through landmark path in a distributed manner. Without global information, GreedyFlow does not try to determine the whole relay path for each packet. Rather, it only selects the next-hop landmark for each packet based on the local information on current carrier to greedily route it to its destination landmark. To realize this goal, GreedyFlow builds a global landmark map and a local traffic map on each node, which represent the node's understanding of the landmark distribution in the network and node transition frequencies between landmarks in the area where the node primarily visits, respectively. The two maps help decide the next-hop landmark for each packet.

Such a design rationale matches with our daily experiences. People usually know the traffic delays on roads connecting places they visit frequently and the general direction to reach a far-away unfamiliar place (e.g., in south or north). Then, people can use such knowledge to greedily relay a message to a far-away place efficiently.

3.2.3. Why not Build a Global Traffic Map

Intuitively, it would be beneficial to have a global traffic map to guide the landmark-by-landmark forwarding. However, this is not adopted since a global traffic map can hardly be maintained accurately in the context of DTN in which nodes usually are sparsely distributed. As a result, the global traffic map cannot guarantee the routing efficiency but incurs much extra overhead. Therefore, we choose to build the local traffic map only. We see later in the experiment that the local traffic map can also support efficient packet routing.

3.3. Global Landmark Map

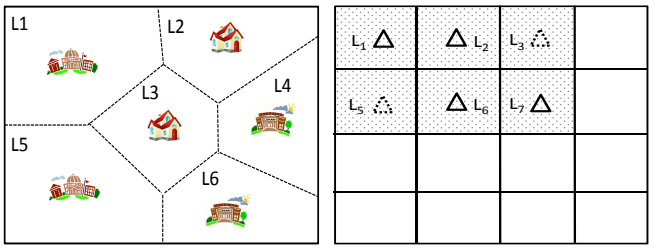
The global landmark map shows the distribution of landmarks in the network. It includes the GPS position of each landmark and the neighboring relationships between landmarks, i.e., the neighbor landmarks of each landmark. It is designed to ensure that packets are forwarded on the right direction towards their destination landmarks.

The global landmark map is generated and maintained by the network administrator. When a DTN is deployed, the administrator selects landmarks in the network. It can collect the mobility information of nodes in the system to determine landmarks. When a node joins in the system, it first obtains the global landmark map from the network administrator. The global landmark map usually remains unchanged for a relative long period of time, which means that the global landmark map on each node does not need frequent updates. When the global landmark map changes, each node can obtain the updated version when it has access to the network administrator, e.g., when moving to a place with network connection.

We split the network into sub-areas based on landmarks and let each landmark be responsible for the sub-area it resides in. The area between two landmarks is evenly split to the two neighboring sub-areas (i.e., the borderline passes through the midpoint of the line connecting the two landmarks and is perpendicular to it), as shown in Figure 2(a). Each sub-area is stored as the list of vertices in clockwise direction. When a node enters the sub-area of a landmark, we regard it as transiting to the landmark. As a result, node mobility can be summarized as consecutive transitions between landmarks.

3.4. Local Traffic Map

Each node maintains a local traffic map to record its knowledge about how frequently nodes transit between landmarks in the area where it primarily visits. It is designed to select the locally optimal landmark path on the direction to the destination landmark for packets carried by the node.



(a) Global landmark map. (b) Coverage of local traffic map.

Figure 2: Example of global landmark map and local traffic map coverage.

3.4.1. Local Traffic Map Coverage

As previously introduced, the local traffic map helps determine the next-hop landmark on the path that can lead to the destination landmark quickly. Therefore, the more landmarks the local traffic map includes, the more likely that the next-hop landmark that can lead to smaller expected delay can be found. However, nodes often have limited storage resources, and the information dissemination often has a long delay in DTNs. This means that a node can neither store the node transit frequencies between all landmarks nor collects such information timely in DTNs. Therefore, we need to determine which landmarks to be included in the local traffic map.

To solve this problem, we first examine how nodes visit landmarks in DTNs. We analyzed three real DTN traces: Dartmouth Campus Trace (DART) [29], DieselNet AP Trace (DNET) [30], and Roma taxi mobility trace (ROMA) [31]. The DART trace shows the association record of the WiFi access points (APs) and students' devices on Dartmouth campus. The DNET trace includes the AP association record of 34 buses in a college town (UMass). The ROMA trace records the GPS position of 320 taxis in the Roma city for one month on 2014. We used the first 10 days of the ROMA trace in this paper for two reasons. First, we want to keep the trace length/size in the same level as the other two traces. Second, the patterns (i.e., measured statistics) identified in the first 10 days of the trace are consistent with the whole trace.

The three traces show DTN scenarios in small (i.e., DNET), medium (i.e., DART), and large (i.e., ROMA) scales with both human mobility (i.e., DART) and vehicle mobility (i.e., DNET and ROMA). Therefore, we believe the results obtained from these traces are representative to a vast amount of DTNs. We preprocessed the three traces to abstract landmarks, i.e., a building or an area with a certain size, and merged neighboring records with the same device and landmark. Finally, the DART trace contains 320 nodes and 159 landmarks, the DNET trace has 34 nodes and 18 landmarks, and the ROMA trace has 320 nodes and 1030 landmarks.

For each trace, we measured the total number of landmarks a node visits and the number of landmarks that account for 70% of a node's landmark visits. The test results with the three traces are shown in Figures 3(a), 3(b), and 3(c), respectively. We ranked nodes in descending order of the two metrics in the two figures. From the three figures, we find that more than 80% of nodes in the DART, DNET, and ROMA traces visit fewer than 20, 5, and 220 landmarks. Besides, most nodes spend their 70% of visits on fewer than 5, 5, and 50 landmarks, respectively, in DART, DNET, and ROMA traces. Such results demonstrate that nodes often only frequently transit between a limited number of landmarks.

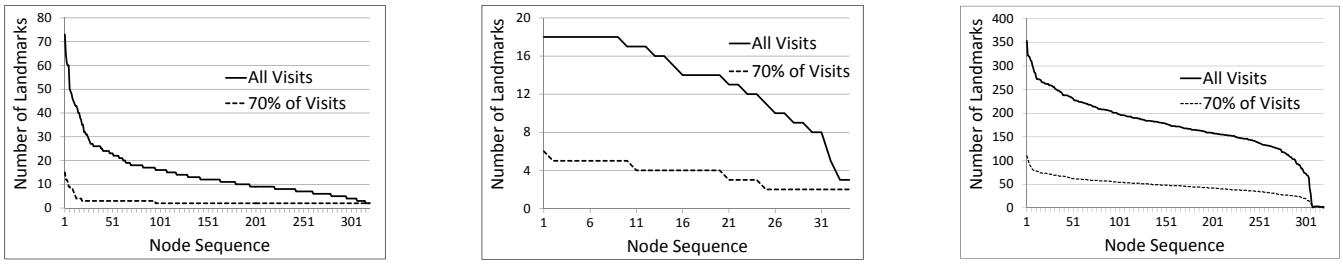
Besides, since DTNs are featured by slow information dissemination, a node cannot timely learn the node transit frequencies between landmarks it rarely visits. Therefore, we can only rely on nodes to forward packets and learn the overall node transit frequencies between landmarks, which is costly and inefficient. The above finding is similar to our daily experiences: people often commute between a few places and are familiar with these places, i.e., knowing the traffic volumes between these places. However, they may not know the traffic volumes in unfamiliar places where they rarely visit.

Therefore, we let the local traffic map of each node only include landmarks in the area where the node primarily visits. Specifically, each node ranks the landmarks in decreasing order of its visit frequencies and selects the first k landmarks that account for V_f% of its total landmark visits. The area covered by these landmarks, i.e., the area covered by the most left-up landmark and the most right-bottom landmark, is defined as the coverage of the local traffic map. Figure 2(b) shows an example of the coverage of a local traffic map. In this example, the primarily visited landmarks are L_1, L_2, L_6, and L_7. Then, the shadowed area is determined as the coverage of the local traffic map, which includes L_1, L_2, L_3, L_5, L_6, and L_7. We can see that the larger V_f is, the more information the local traffic map provides, but also the more overhead incurred. Therefore, a suitable V_f can be determined based on the requirement on routing efficiency and overhead. Based on empirical measurement, we set v_f to 70% in this paper.

3.4.2. Local Traffic Map Construction

Each node updates the local traffic map upon encountering other nodes. Specifically, suppose the coverage of a node's local traffic map is $C_i = \{L_a, L_b, L_c, L_d\}$, $\{a, b, c, d\} \in [1, m]$, it queries each encountered node about how frequently it transits between these landmarks, i.e., from L_x to L_y, $x, y \in \{a, b, c, d\}$ and $x \neq y$. To enable such a function, each node builds an individual landmark transit table to record its transit frequencies between landmarks, as shown in Table 1. Each row represents the node's transit frequency (i.e., how many transits per day) between two neighboring landmarks. The third column and the fourth column are the transit frequencies from L_x to L_y and from L_y to L_x, respectively.

Each node collects encountered nodes' transit frequencies to update its local traffic map. For example, suppose a node's local traffic map contains landmark L_1 and landmark L_2. Then, to get the overall transit frequency from



(a) DART trace. (b) DNET trace. (c) ROMA trace.

Figure 3: Distribution of the # of visited landmarks.

Table 1: Landmark transit table.

$Landmark_x$	$Landmark_y$	$Frequency_{xy}$	$Frequency_{yx}$
L_1	L_5	3	2.5
L_2	L_{13}	8	9
L_{15}	L_{24}	7	4
...	...	...	...

L_1 to L_2, the node gets every encountered node's transit frequency from L_1 to L_2 and sums up these transit frequencies. However, in this process, a node may meet another node and obtain its transit frequency multiple times. Then, to avoid summing up a node's transit frequency multiple times, each node maintains a record on which nodes' transit frequencies have already been included in the calculation of the overall transit frequency from one landmark to another. We name such a record as the transit element table. Table 2 shows an example of transit element table on a node for the transition from L_1 to L_2.

Table 2: Transit element table.

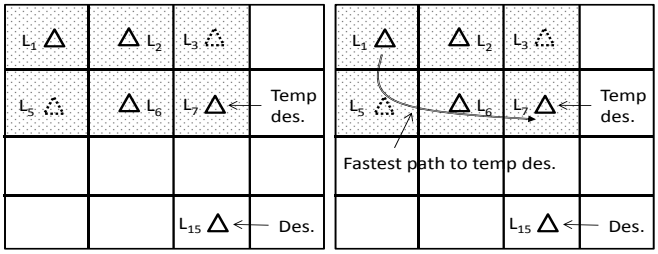
Transit	Node	Frequency
$L_1 \rightarrow L_2$	N_8	0.7
	N_5	2
	N_7	1.5
	...	...
	Overall	10.5

The "overall frequency" in each transit element table finally is stored in the local traffic map. Table 3 shows the traffic map following the example in Figure 2(b).

Table 3: Local traffic map.

Landmark ID	Neighbor Landmark	Frequency
L_1	L_2	10.5
	L_5	13
L_2	L_1	9
	L_6	4
	L_3	10.2
L_3	L_2	4.2
	L_7	5.5
...	...	...

In summary, in GreedyFlow, when a node, say N_i, meets another node, say N_j, N_i obtains N_j's transit frequencies between each pair of landmarks covered in N_i's



(a) Temporary destination land-(b) Fastest path to the temp des. mark.

Figure 4: Determining the next-hop landmark.

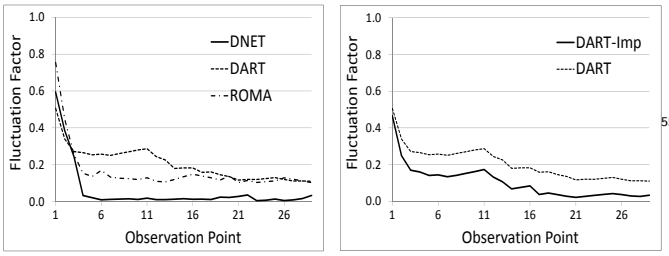
local traffic map to update its local traffic map. In detail, for $L_x \rightarrow L_y$, if N_j's transit frequency (denoted by f_{xy}^j) is not 0, N_i first checks whether it has a transit element table for this transit. If not, it creates a transit element table for this transit with only one entry, i.e., the entry for N_j. N_i then updates N_j's entry in the transit element map to f_{xy}^j. N_i also updates the overall frequency for transit $L_x \rightarrow L_y$ accordingly in both the transit element table and the local traffic map.

For example, suppose a node's transit element table for $L_1 \rightarrow L_2$ is as shown in Table 2, and its local traffic map is as shown in Table 3. When the node meets N_8 and finds that N_8's transit frequency for $L_1 \rightarrow L_2$ has changed to 1.4, it first updates the entry for N_8 accordingly. Then, it updates the overall transit frequency for $L_1 \rightarrow L_2$ to 11.2 in both the transit element table and the local traffic map.

3.4.3. Consistency of Local Traffic Maps

As introduced in the previous section, each node builds its local traffic map independently. Since nodes may meet different sets of nodes, the transit frequencies between the same pair of neighboring landmark maintained in the local traffic maps of different nodes may be different. However, as mentioned later in Section 3.5, each node makes packet forwarding decisions sorely based on its local traffic map. This means that the inconsistency in the local traffic maps may potentially deteriorate the packet routing efficiency.

Therefore, we measure the consistency of local traffic maps with the three real traces to show how current design performs. Specifically, for each landmark hop, e.g., L_x to L_y, we collect its overall transit frequency stored in all local traffic maps that contain this landmark hop and



(a) Inconsistency among local traffic maps with the three traces.

(b) Improvement with the advanced method on the DART trace.

Figure 5: Illustration of inconsistency and improvement.

calculate the standard deviation of all transit frequencies. To reflect the significance of the deviation, we further divide the standard deviation by the average frequency, i.e., *deviation/average*, and denote such a value as the fluctuation factor. Finally, we calculate the average fluctuation factor of all landmark hops to show the consistency of local traffic maps. We measured the average fluctuation factor at 29 evenly distributed observation points and showed the results in Figure 5(a).

We see from the figure that with all the three traces, the fluctuation factor is large in the beginning and decreases as more trace data is analyzed. This is because, as time goes one, different local traffic maps converge when each node meets more nodes. However, the fluctuation factor in the DART and ROMA trace is still high ($> 10\%$) even at the end of the trace, while that in the DNET trace becomes very small quickly ($< 3\%$). This is because both DART and ROMA represent a much larger scenario than the DNET trace in which nodes are more distributively distributed. Therefore, even when each node can collect more data, the inconsistency among local traffic maps is still obvious in the two traces. Such a result shows that the inconsistency among local traffic maps may not always be well controlled.

3.4.4. Enhancing the Consistency of Local Traffic Maps

The results in the previous section show that inconsistency commonly exists in the local traffic maps generated through the default method introduced in Section 3.4.2. Therefore, we propose an advanced local traffic map update method to enhance the consistency between local traffic maps. In this method, when two nodes meet, each node contributes not only its own transit frequencies but also the records in its transit element tables to the other node. Specifically, we expand the transit element table to include the most recent update time of the transit frequency of each node, as shown in Table 4. Then, when two nodes meet, they first update the advanced transit element table following the same way as introduced in Section 3.4.2. After this, they merge their advanced transit element tables for all common transits in their local traffic maps with the following rules. For each common transit, 1) if the transit frequency of a node exists in the advance transit element table of both nodes, the one with the more recent update time is kept in both nodes. 2) If the transit frequency of a node only exists in the advance transit element table of one node, it is copied to the element table of the other node with associated update time.

Table 4: Advanced transit element table.

Transit	Node	Frequency	Update Time
$L_1 \rightarrow L_2$	N_8	0.7	10
	N_5	2	40
	N_7	1.5	100
	...	...	...
	Overall	10.5	100

For example, suppose when N_i and N_j meet, they both have the landmark hop $L_1 \rightarrow L_2$ in the local traffic map. Suppose the advanced transit element table for $L_1 \rightarrow L_2$ on N_i includes the transit frequencies of N_1, N_4, and N_9 with update time $10s$, $15s$, and $30s$, respectively. We also suppose the advanced transit element table for $L_1 \rightarrow L_2$ on N_j includes the transit frequencies of N_4, N_9, and N_{17} with update time $11s$, $35s$, and $40s$, respectively. Then, the final merged advanced transit element table on both nodes will include transit frequencies of N_1, N_4, N_9, and N_{17} with update time $10s$, $15s$, $35s$, and $40s$, respectively. The basic rule is that the one with the more recent change or only exists in one node is kept.

With the above steps, the two nodes converge on the advanced transit element table for common transits in their local traffic maps. Consequently, different nodes will maintain better consistency on their local traffic maps. We further evaluate the performance of such an advanced method following the same metric in Section 3.4.3. Since the DART trace presents the most inconsistency previously, we focus on it in this test. The result is shown in Figure 5(b), in which DART-Imp demotes the result with the advanced method.

We see from the figure that when the advanced local traffic map update method is enabled, the "Fluctuation Factor" is reduced by almost half in the beginning and reaches almost 0 after half of the observation points in the DART trace. This shows that the advanced update method owns the ability to effectively improve the consistency among local traffic maps. However, such an improvement comes at additional communication costs. Instead of only exchanging each node's transit frequencies in the baseline method (introduced in Section 3.4.2), two encountered nodes now have to exchange their advanced transit element tables for all shared links in their local traffic maps. Fortunately, the size of the local traffic map of a node often is limited (i.e., Section 3.4.1), and the amount of shared links may only account for partial of the map. This avoids the additional communication overhead to become uncontrollable. Consequently, the advanced update method is more suitable in DTNs in which nodes own relatively abundant communication resources.

Furthermore, even with abundance resources, it would

be a waste if the advanced method is enabled blindly. This is because some DTNs may lead to a good consistency between the local traffic maps even with the baseline scheme introduced in Section II-D2. The DNET trace is such an example, as shown in Figure 5(a). Therefore, to handle this issue, we further propose the following scheme to better balance the overhead and benefit.

- When two nodes meet, they follow the advanced method to update their local traffic map with a probability of $P_u \in [0, 1]$. The value of P_u is proportional to the inconsistency (i.e., fluctuation factor) observed in the past update.
- As a result, the advanced method is more frequently used when there is a large inconsistency among local traffic maps and less frequently otherwise. This makes sure that the cost is paid off by the benefit.

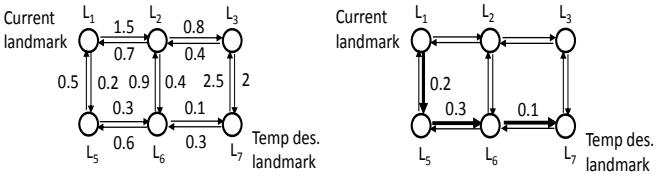
3.5. Packet Routing in GreedyFlow

We introduce the packet routing algorithm in this section. We first give out the overview and then the details.

3.5.1. Overview

The packet routing works in a greedy manner. When a node generates or receives a packet, it checks its local traffic map and the global landmark map to decide the next-hop landmark for the packet. Specifically, since a node's local traffic map may not include the destination landmark of the packet, it first selects a temporary destination landmark in the local traffic map that has the closest distance to the destination landmark, which is calculated with the information from the global landmark map. This ensures that the packet forwarding is always on the right direction. We introduce this step in Section 3.5.2. Then, the node finds the fastest path to the temporary destination landmark based on the local traffic map and as well as the next-hop landmark on the path. We introduce how to determine such a path in Section 3.5.3. Figure 4(b) shows that the fastest path from L_1 to temporary destination landmark L_7 is $L_1 \rightarrow L_5 \rightarrow L_6 \rightarrow L_7$ and the next-hop landmark is L_5.

Then, the packet is forwarded to the selected next-hop landmark L_5. In detail, the node queries its neighbors about where they are predicted to move to and forwards the packet to the node that is predicted to move to the selected next-hop landmark L_5. The details on how each node predicts the next landmark it is going to transit to is introduced in Section 3.5.4. When the node arrives at L_5, it repeats the above process to further forward the packet. Finally, the packet is greedily forwarded toward its destination landmark. However, a node that is predicted to move to the next-hop landmark may not always be found. A node may not always move to the predicted landmark. A packet may revisit the same landmark during the forwarding. We discuss how to handle these exceptions in Section 3.5.5.



(a) Abstracted local traffic map. (b) The selected fastest path.

Figure 6: Determine the fastest path to the temporary destination.

3.5.2. Selecting Temporary Destination Landmark

The packet carrier first checks whether its local traffic map contains the destination landmark of the packet or not. If yes, the temporary destination landmark is the destination landmark. Otherwise, the temporary destination landmark is the landmark in the local traffic map that has the closest distance to the destination landmark. The distance between two landmarks is calculated as the minimal number of landmark hops between them. As shown in Figure 4(a), suppose the destination landmark is L_{15}, landmark L_7 is selected as the temporary destination landmark since it has the closest distance to L_{15}.

3.5.3. Determine the Fastest Path to the Temporary Destination

As introduced in Section 3.2, the expected delay to a forwarding hop is determined by the frequency of node transition on the hop. We use such a property to find the fastest path to the temporary destination landmark.

For better illustration, we abstract each landmark as a circle. Two circles are connected if are neighbors. We also abstract the node transition from one landmark to another landmark as a link connecting the two landmarks. Therefore, two neighbor circles are connected by two links in two direction. Each link has a weight, which represents the delay to forward a packet through the link. It is calculated as $1/f_t$, where f_t is the frequency of node transition in the direction of the link. Figure 6(a) shows the abstracted local traffic map in which each link has a weight. Note that there are two weighted links (one for each direction) between each pair of neighbor landmarks. This is because the frequency of node transitions in different directions may be different.

Using such a graph, the problem of finding the fastest path to the temporary destination landmark turns into the problem of finding the shortest path to the temporary destination landmark. We use the Dijkstra algorithm [32] to fulfill this task. In detail, we take the current landmark L_1 as the root and iteratively find the shortest path to the temporary destination landmark L_7, i.e., the bold lines in Figure 6(b) (i.e., $L_1 \rightarrow L_5 \rightarrow L_6 \rightarrow L_7$). Then, the next landmark after current landmark on the shortest path, i.e., L_5, is identified as the next-hop landmark.

Note that the local traffic map is not static. As introduced in Section 3.4.2, each node updates its local traffic map upon collecting the landmark transit information from encountered nodes. Thus, the change on the path

weights will be reflected in the local traffic map. Then, the Dijkstra based path selection algorithm will be able to identify the best path dynamically, even when it is not the physically shortest one.

3.5.4. Predicting Node Mobility

In order to forward a packet from current landmark to the determined next-hop landmark, the current packet holder queries its neighbors about where they are most likely to transit to and forwards the packet to the node that is going to move to the next-hop landmark. To fulfill this function, each node predicts its next transit upon moving to a new landmark. In detail, each node uses its historical landmark visit information to feed the Order-1 Markov predictor to deduce the landmark it is going to transit to. In detail, suppose a node's landmark visit history can be represented by $V_H = L_{x_1}L_{x_2}L_{x_3}\cdots L_{x_{n-1}}L_{x_n}$. Then, the probability that the node is going to visit landmark $L_{x_{n+1}}$ can be calculated by

$$Pr(L_{x_n}L_{x_{n+1}}|L_{x_{n-1}}L_{x_n}) = \frac{Pr(L_{x_{n-1}}L_{x_n}L_{x_{n+1}})}{Pr(L_{x_{n-1}}L_{x_n})}, \quad (1)$$

where

$$Pr(L_{x_{n-1}}L_{x_n}L_{x_{n+1}}) = \frac{N(L_{x_{n-1}}L_{x_n}L_{x_{n+1}})}{N(All_3)} \quad (2)$$

and

$$Pr(L_{x_{n-1}}L_{x_n}) = \frac{N(L_{x_{n-1}}L_{x_n})}{N(All_2)} \quad (3)$$

where $N(L_{x_{n-1}}L_{x_n}L_{x_{n+1}})$ is the number of times that the node visits landmarks $L_{x_{n-1}}$, L_{x_n}, and $L_{x_{n+1}}$ consecutively, $N(L_{x_{n-1}}L_{x_n})$ denotes the number of times that the node visits landmarks $L_{x_{n-1}}$ and L_{x_n} consecutively, and $N(All_k)$ $(k = 2, 3)$ means the number of visits on consecutive k landmarks. Finally, the landmark that leads to the largest transit probability in Equation (1) is selected as the landmark that the node is going to move to.

3.5.5. Handle Exceptions

As mentioned in previous sections, GreedyFlow relies on predicted node mobility to forward a packet from current landmark to the next-hop landmark. However, such predictions may not always be correct. As a result, packet forwarding may face three exceptions. Firstly, after determining the next-hop landmark for a packet, the current packet carrier may not be able to find a neighbor node that is predicted to move to that landmark. Secondly, even when such a node is found, it may actually moves to a landmark that is different from the predicted one, making the packet deviated from the planned forwarding path. Thirdly, due to prediction errors, a packet may return to a landmark that it has already visited. We then design three additional schemes to handle the three exception cases.

When the first exception case happens, we simply let the current carrier of the packet continue carrying it until arriving at another landmark. If this landmark is the next-hop landmark of the packet, the exception is solved automatically. If not, the first exception turns into the second exception, i.e., the packet carrier moves to an unexpected landmark, the handling of which is discussed in the next paragraph.

When the second exception happens, the current carrier first checks whether it can find a node from neighbor nodes that is predicted to move to the expected next-hop landmark of the packet. If yes, the packet is forwarded to the node. This node is expected to carry the packet to the next-hop landmark, thus resuming the planned forwarding path. If no such nodes can be found, the packet is forwarded to the neighbor node that has the highest centrality. Such a node starts over the packet forwarding process from the current landmark by following the procedures presented in Sections 3.5.2, 3.5.3,and 3.5.4 to handle the packet. The centrality of a node is defined as the number of nodes it can meet in a unit time. We select such a node to carry the packet since it can meet more nodes and thus has more options on potential packet forwarders.

Although GreedyFlow uses the global landmark map to determine the packet forwarding direction, the third exception can still happen. The reason is the same as that for the second exception, i.e., the node movement prediction may not always be correct. When such an exception happens, we do not try to resume the planned forwarding path but let the current packet carrier start over the packet handling process directly (i.e., trying to select a new path for the packet). This is because such an exception means that nodes' movement to the previous next-hop landmark is hard to predict.

3.5.6. Summary

We summarize the process of packet routing in GreedyFlow as follows. We also show the pseudo-code of the case when a node tries to forward a packet in Algorithm 1.

- When a node generates a packet or carries a packet to its next-hop landmark, the node follows the method introduced in Section 3.5.2 to determine its temporary destination landmark.
- The node determines the fastest path to the temporary destination landmark based on its local traffic map and selects the next-hop landmark for the packet by following the method in Section 3.5.3
- Then, the node checks whether a neighbor node is predicted to move to the next-hop landmark. If yes, it forwards the packet to the neighbor node. The prediction of node transition is introduced in Section 3.5.4.
- If no suitable carrier can be found or the selected carrier moves to a landmark other than the next-hop landmark, the packet is handled by the schemes in Section 3.5.5.
- The above process repeats until the packet arrives at the destination landmark.

Algorithm 1 Pseudo-code of the GreedyFlow routing when node N_i tries to forward a packet p destined to landmark L_d.

```
1: procedure EXCHANGETOPFRIENDSWITH(b)
2:     N_i.determineTemporaryDst(p)
3:     N_i.selectNextHopLM(p)
4:     N_i.forwardPacket(p)
5: end procedure
6: procedure DETERMINETEMPORARYDST(p)
7:     mindist = max;
8:     for each landmark l in local traffic map do
9:         if dist(l, L_d) < mindist then
10:            mindist = dist(l, L_d);
11:            tempDst = l;
12:        end if
13:    end for
14: end procedure
15: procedure SELECTNEXTHOPLM(p)
16:    path = findFastestPath(tempDst);
17:    L_next = path.returnNextLandmark() ;
18: end procedure
19: procedure FORWARDPACKET(p)
20:    forwarder = N_i;
21:    prob = getProb(N_i, L_next); //calculate N_i 's probability to
       transit to the L_next
22:    for each neighbor node n in current landmark do
23:        if prob < getProb(n, L_next) then
24:            forwarder = n
25:            prob = getProb(n, L_next)
26:        end if
27:    end for
28:    forward(p, forwarder);
29: end procedure
```

3.6. Cost and Applicability Analysis

We analyze the cost of the GreedyFlow in terms of storage, communication, and computing in this subsection. We also discuss the applicability of GreedyFlow and the tradeoff on routing performance and cost.

3.6.1. Storage Cost

According to the design in Section 3.3 and 3.4, the information that has to be stored in each node under GreedyFlow includes a global landmark map (e.g., Figure 2(a)), a local traffic map (e.g., Table 3), a transit element table (e.g., Table 2), and a landmark transit map (e.g., Table 1). The four maps/tables store the locations and neighbor relationships of all landmarks, the overall transit frequencies within a node's local traffic map, individual nodes' transit frequencies within a node's local traffic map, and a node's transit frequencies among all landmarks, respectively.

Therefore, suppose there are m landmarks in the DTN, the four tables each has a storage cost of $m*\overline{G}*\overline{S}$, where $\overline{G}$ is the average amount of neighbor landmarks a landmark has (e.g., 3 or 4), and $\overline{S}$ represents the average size of the recorded data between a pair of neighbor landmarks in a table (which can be up to 0.5 KB). This makes GreedyFlow own a higher storage cost than probabilistic DTN routing algorithms [15, 16, 17, 18] that require nodes to only record the encountering frequencies with others. However, the absolute storage cost of a table in a large DTN with 1000 landmarks is around a few megabytes. We believe this is affordable for nowadays' mobile devices.

3.6.2. Communication Cost

The communication between encountered nodes serves two functions in GreedyFlow: update the local traffic map (Section 3.4.2) and conduct packet forwarding (Section 3.5.1). For the traffic map update, two encountered nodes exchange their transit frequencies between landmarks in their local traffic maps. For the packet forwarding, two encountered nodes exchange their predicted next-hop landmark and packets destined for the other node. Thus, the overall communication cost in an encountering can be represented as $m*S_t+S_n+S_p$, where m is the number of landmarks, S_t denotes the size of the transit frequency data between two landmarks, and S_n is the size of the next-hop landmark ID, and S_p is the size of packets.

We see that GreedyFlow does not incur excessive communication overhead when compared with probabilistic routing algorithms as S_t and S_n often are a few bytes, and S_p is standard for all DTN routing algorithms. This makes GreedyFlow resilient to intermittent contacts in DTNs.

3.6.3. Computing Cost

GreedyFlow introduces computing costs in updating the local traffic map (Section 3.4.2) and conducting packet forwarding (Section 3.5.1). The former just updates the transit frequencies of neighbor landmark pairs and thus incurs a computing cost of $O(m)$. The latter mainly does two tasks: 1) finds the temporal destination landmark following the Dijkstra algorithm in the local traffic map (Section 3.5.3); and 2) calculate the probability of transiting to other landmarks (Section 3.5.4). The two tasks have a computing cost of $O(\overline{m}^2)$ and $O(m)$, respectively, where $\overline{m}$ is the average number of landmarks in a local traffic map, and m is the total number of landmarks.

Consequently, the overall computing cost is $O(\overline{m}^2+m)$. Since a local traffic landmark only includes landmarks that a node primarily visits, $\overline{m}$ often is small. This shows that the computing cost is acceptable to current mobile devices.

3.6.4. System Applicability

The above analysis shows that GreedyFlow presents a high storage cost and a normal communication and computing cost, when compared with current probabilistic DTN routing algorithms. However, storage is not the major challenge for mobile devices currently. Actually, nodes in DTNs are challenged by limited power and communication opportunities, which raises the need of controlling the computing and communication cost of the routing algorithm. The analysis results illustrate that GreedyFlow satisfies this requirement.

Therefore, GreedyFlow is best applied in DTNs in which nodes own sufficient storage resources. Actually, there is a tradeoff on routing performance and the cost (applicability). Particularly, to extend the applicability of GreedyFlow, the communication and computing cost has

been carefully controlled in current design, which limits the routing performance. When there is more communication/computing resource, the routing efficiency could be further enhanced. For example, the advanced component in Section 3.4.4 could improve the routing efficiency at the cost of more communication overhead. The accuracy of the prediction of the next-hop landmark in Section 3.5.4 can also be enhanced with more complex algorithms. We leave the research on how to optimally balance or flexibly tradeoff the two goals to future work.

3.7. Advanced Enhancement on Packet Routing

We further design one optional advanced enhancement that can improve the packet routing efficiency at additional costs. As introduced in Section 3.2, GreedyFlow relies on the landmark-by-landmark path to forward a packet to its destination landmark. In this process, node mobility is modeled and used as consecutive transits between landmarks. Actually, as revealed in literatures [5, 6, 7] and our investigation (introduced in Section 3.4.1), if we look at node mobility from the perspective of visiting landmarks, nodes usually visit a few landmarks frequently. This is the basis of node-based forwarding strategy. However, as introduced in Section 3.2, purely relying on such a feature can lead to limited routing efficiency since the number of nodes that can frequently visit a landmark often is limited.

However, this does not mean that node-based forwarding has no value. In case nodes that can frequently visit the destination landmark are found (though the number of such nodes may be small), we should take advantage of it to further enhance the routing performance. We then propose an advanced component to effectively synergize the landmark-based forwarding (as proposed earlier) and node-based forwarding. Generally, the landmark based scheme is adopted first to gradually forward a packet towards its designation landmark. In this process, whenever a node that is not overloaded and can bring the packet to its destination landmark faster than following the landmark based path is encountered, the packet is forwarded to the node for node-based forwarding. As a result, packet routing efficiency can be further enhanced. We carefully design the requirements for such a switch to happen, thereby guaranteeing the benefits.

3.7.1. Maintain Visiting Frequencies

In order to support the advanced enhancement, each node needs to maintain its visiting frequencies with landmarks. The visiting frequencies also decay over time to reflect the change of a node's landmark visiting pattern. Specifically, node N_i's visiting frequency with landmark L_x (denoted V_{ix}) is updated every T_f seconds by the following formula.

$$V_{ix} = \alpha * V_{ix} + (1 - \alpha) * M_{ix}/T_f$$

where α is a decay factor and M_{ix} denotes the number of times that N_i visits L_x in the past T_f seconds. To reflect the importance of recent visits, α is set to 0.6 and T_f is set to 18000 in this paper. They can be adjusted to fit the requirements in different DTN scenarios.

3.7.2. Delay Estimation

To optimally determine whether a packet should be forwarded to a node that can frequently visit its destination landmark (i.e. node-based forwarding) or continue be forwarded through the landmark-based path (landmark-based forwarding), we need to estimate the expected delay through the two ways. The estimated delay for the former case (i.e., node-based forwarding) is calculated as $1/V_{ix}$, where V_{ix} denotes the node's visiting frequency to the destination landmark L_x.

For the latter case (i.e., landmark-based forwarding), we can only calculate the expected delay to reach the temporary destination (introduced in Section 3.5.2). This is because we can only obtain the transit frequencies between landmarks in the local traffic map. However, this information is still useful for us to make a decision, which will be introduced in the next subsection. Such a delay is calculated by summing up the expected delay on each hop on the landmark based path. Note that the weight of the link representing a hop introduced in Section 3.5.3 (i.e., 1 over the overall transit frequency on the hop) actually represents the delay on this hop.

Note that though the above delay estimation is not deterministic, the benefit is guaranteed on average as long as node mobility is not completely random. Kindly note that forwarding decisions generally are made on the expected possibility/delay of delivery in the context of DTN with opportunistic mobility (not scheduled mobility). Such a rationale is followed by all existing probabilistic/social network based DTN routing protocols mentioned in the related work.

3.7.3. Switching to the Node-based Forwarding

With the advanced enhancement, the handling of a packet turns to the node-based forwarding when it can deliver the packet faster. Specifically, the node-based forwarding is adopted when the following two requirements are satisfied.

- The current packet carrier meets a node that can carry the packet to the destination landmark faster than forwarding the packet through the landmark based path.
- The packet buffer of the node is not full.

The first requirement means that the expected delay when the packet is forwarded by the node, denoted D_n, is smaller than that when the packet is forwarded through the landmark based path, denoted D_d. As mentioned in the previous section, we can hardly estimate the delay of a packet when it is forwarded through the landmark based path due to the limitation of the local traffic map. We

therefore use the expected delay to the temporary destination landmark (denoted D_{td}), which can be easily deduced based on the local traffic map, to check whether the first requirement is satisfied. This is because D_{td} must be smaller than or equal to D_d. Then, as long as $D_n < D_{td}$, we can have $D_n < D_d$, which means that the first requirement is satisfied. The second requirement is simply used to prevent overloading nodes.

It is true that using D_{td} in evaluating the first requirement may miss some cases when D_n is larger than D_{td} but smaller than D_d. However, we argue that such a design brings about certain benefits. First, this limits the frequency that the node-based forwarding is adopted, i.e., it is used when it clearly shows lower expected delay than the landmark-based forwarding. This follows our motivation in the introduction that node-based forwarding cannot fully utilize node mobility. Second, D_{td} usually is smaller than D_n when the packet is far away from the destination landmark. This is because, in this case, the packet's distance to the temporary destination is much shorter than the distance to the actual destination (since the local traffic map often is small). As a result, node-based forwarding is adopted mainly when the packet is close to its destination, which follows our design rationale.

4. Performance Evaluation

We first evaluate the performance of GreedyFlow without advanced schemes in comparison three state-of-art methods. We then evaluate the enhancement of the two advanced schemes, which are proposed in Section 3.4.4 (i.e., advanced local traffic map update) and Section 3.7 (i.e., advanced packet routing), in section 4.4.

4.1. Experiment Settings

We conducted event driven experiments with the three real traces, namely Dartmouth campus trace (DART) [29], DieselNet AP trace (DNET) [30], and Roma taxi mobility trace (ROMA) [31]. Section 3.4.1 introduces the details of the three traces. We adopted three representative comparison algorithms: Geomob [12], PER [33] and SimBet [34]. Geomob is similar to GreedyFlow that it also routes packets in a landmark-by-landmark manner. However, it requires that all nodes know the traffic distribution in the network to decide the landmark path for each packet. As mentioned in the introduction, such a requirement is not practical. We still use it to measure whether GreedyFlow can lead to comparable performance with Geomob. PER estimates each node's probabilities to visit each landmark and forwards packets to nodes that have a high probability to visit their destination landmarks before they expire. SimBet evaluates a node's suitability to carry a packet by considering both its centrality and its visit frequency with the packet's destination landmark.

In the experiment, we used the first 1/3 of testing traces for initialization, in which nodes build the local traffic map in GreedyFlow and accumulate related metrics, e.g., landmark visit frequencies, in comparison methods. After this step, packets are generated with randomly selected destination landmarks at the rate of r_p packets per landmark per day. The TTL (Time to Live) of each packet was set to 30 days, 6 days, and 2 days in the DART, DNET, and ROMA trace, respectively. When a packet's TTL expires, it is dropped directly. We assume that each packet has the size of 1 KB and the available storage on each node is m_n KB. We set r_p and m_n to 50 and 150 by default and varied them for extensive performance evaluation. When the storage on a node is full, the oldest packet is dropped. We set V_f to 70 since we find that it is sufficient for efficient packet routing. We set the confidence interval to 95%.

We used four metrics in the experiments: success rate, average delay, forwarding cost, and maintenance cost. The success rate refers to the percentage of packets that are successfully delivered within the TTL. The average delay refers to the average delay of successfully delivered packets. The forwarding cost refers to the number of packet forwarding operations. The maintenance cost refers to the number of messages exchanged between nodes to support packet routing (e.g., transit information in GreedyFlow and landmark visit frequencies in comparison methods).

4.2. Performance with Different Packet Rates

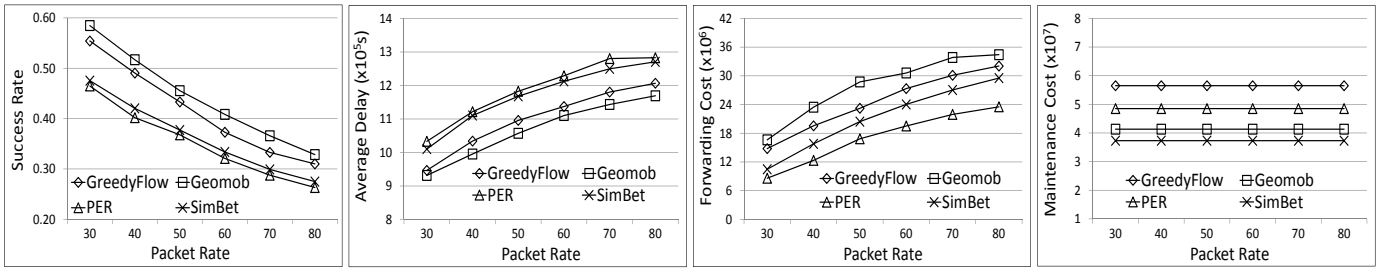
We first conduct performance evaluation with different packet rates (r_p). We varied r_p from 30 to 80 in the test.

4.2.1. Success Rate

Figures 7(a), 8(a), and 9(a) show the success rates of the four methods in the experiments with different packet rates using the DART, DNET, and ROMA trace, respectively. We see from the three figures that the success rates follow: $Geomob \approx GreedyFlow > SimBet \approx PER$.

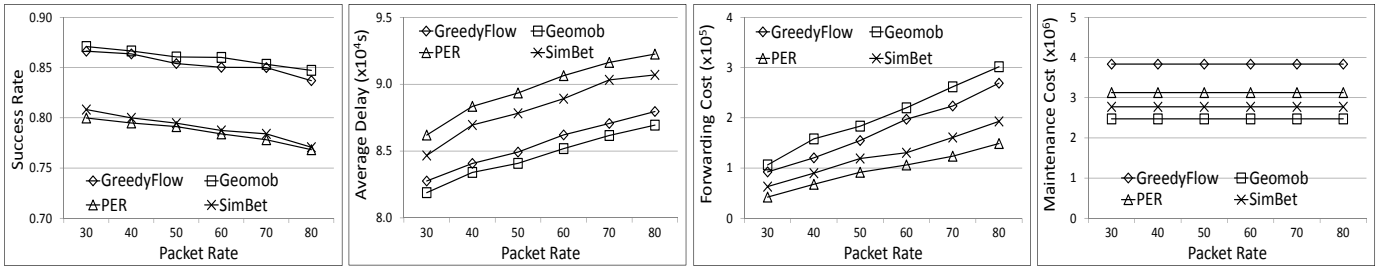
Geomob and GreedyFlow lead to higher success rate than the other two methods because they forward packets in a landmark-by-landmark manner, thereby better utilizing node mobility for more efficient packet routing. Such a result demonstrates the advantage of the landmark path based routing strategy in routing packets to different landmarks. GreedyFlow shows slightly lower success rate than Geomob. This is because nodes in Geomob are assumed to know the global traffic distribution beforehand, while nodes in GreedyFlow only know the traffic distribution in the area they often visit. However, the assumption in Geomob is not practical in real DTNs. We see that the difference on success rate is very marginal. This means that GreedyFlow can also effectively select a fast landmark path for each packet even without the global information. In other words, the performance of the distributed GreedyFlow is comparable to that of Geomob.

We also find that SimBet shows slightly higher success rate than PER. This is because SimBet considers not only a node's visiting frequency with the destination landmark



(a) Success rate. (b) Average delay. (c) Forwarding cost. (d) Maintenance cost.

Figure 7: Performance with different packet rates using the DART trace.



(a) Success rate. (b) Average delay. (c) Forwarding cost. (d) Maintenance cost.

Figure 8: Performance with different packet rates using the DNET trace.

but also its centrality in the network, while PER only considers the visit frequency. Then, in addition to nodes that frequently visit packet destinations, SimBet also utilizes nodes with a high centrality to route packets. As a result, SimBet utilizes more mobile nodes for routing, leading to higher success rate.

4.2.2. Average Delay

Figures 7(b), 8(b), and 9(b) show the average delays of the four methods in the experiments with different packet rates using the DART, DNET, and ROMA trace, respectively. We find from the three figures that the average delays follow: $Geomob<GreedyFlow<SimBet<PER$.

Geomob has the least average delay because each node knows the global traffic distribution, which enables it to always select the landmark path with the minimal expected delay for each packet. In GreedyFlow, each node only knows the node transit frequencies between landmarks in the area where it primarily visits. Therefore, it has slightly higher average delay than Geomob. However, we see that the difference is very small. Since GreedyFlow does not require the global traffic information as Geomob, it is more suitable for DTNs.

SimBet and PER exhibit much higher average delay than Geomob and GreedyFlow. This is because they rely on nodes that frequently visit destination landmarks for packet routing, and such nodes may not always exist. On the contrary, Geomob and GreedyFlow forward packets in the landmark-by-landmark manner to reach their destinations. Nodes that rarely visit the destination landmark of a packet can still be utilized to forward it to a landmark closer to the destination landmark, leading to a small average delay of successfully delivered packets.

4.2.3. Forwarding Cost

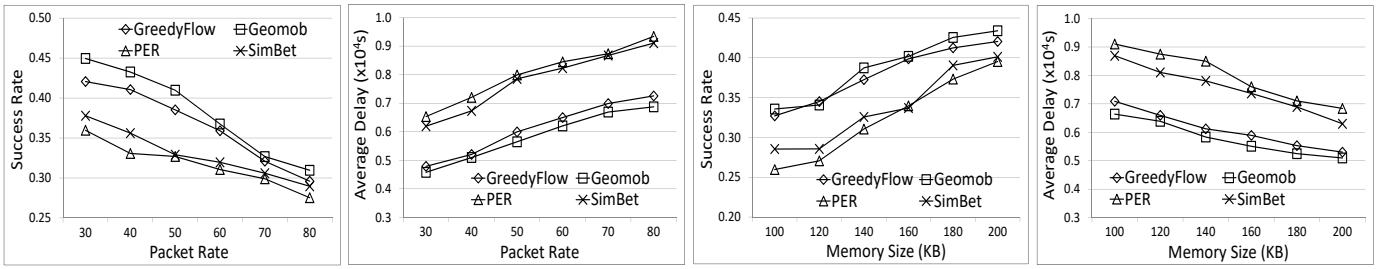
Figures 7(c) and 8(c) show the forwarding costs of the four methods under different packet rates using the DART trace and the DNET trace, respectively. The result with the ROMA trace is similar and thus is not shown. We find from the two figures that the forwarding costs follow: $Geomob>GreedyFlow>SimBet>PER$.

PER generates the least packet forwarding cost because it only forwards a packet to the node that has a high probability of delivering the packet before it expires, leading to few forwarding opportunities. SimBet works in a similar manner as PER but additionally considers centrality for forwarder selection, resulting in more packet forwarding than PER.

GreedyFlow and Geomob generate more packet forwarding than PER and SimBet because they forward packets in a landmark-by-landmark manner, which exploits more nodes to carry packets. However, we can see that the amount of increase is not significant, which is worthwhile considering the improvement on routing efficiency.

4.2.4. Maintenance Cost

Figures 7(d) and 8(d) show the maintenance costs of the four methods under different packet rates using the DART trace and the DNET trace, respectively. The result with the ROMA trace is similar and thus is not shown. We see from the two figures that the maintenance costs follow: $GreedyFlow>SimBet>PER>Geomob$.



(a) Success rate under packet rates. (b) Average delay under packet rates. (c) Success rate under memory sizes. (d) Average delay under memory sizes.

Figure 9: Performance with different packet rates/memory sizes using the ROMA trace.

GeoMob generates the least maintenance cost because it assumes that nodes already know the global traffic distribution beforehand. Therefore, nodes only need to exchange their probabilities of going to neighbor landmarks to support packet routing. In PER, encountering nodes exchange their probabilities to visit all landmarks to determine packet forwarders, leading to more maintenance cost than Geomob. In addition to landmark visit frequencies, nodes in SimBet also exchange centrality information. Therefore, it has higher maintenance cost than PER. GreedyFlow has more maintenance cost than others because nodes need to exchange transit frequencies for local traffic map update. However, we see that the maintenance cost of GreedyFlow is on the same level with others.

Combining all above results, we conclude that the two methods that forward packets through landmark paths lead to better performance than other methods. This justifies the correctness of such a packet routing strategy. We also see that GreedyFlow shows close performance with Geomob, which requires global traffic distribution. Such a result demonstrates that GreedyFlow can realize efficient packet routing in a fully distributed manner.

4.3. Performance with Different Memory Sizes

We further evaluate the performance of the four methods with different memory sizes on each node (m_n). We varied m_n from 100 to 200 in the test.

4.3.1. Success Rate

Figures 10(a), 11(a), and 9(c) illustrate the success rates of the four methods in the experiments with different memory sizes using the DART, DNET, and ROMA trace, respectively. We see from the three figures that the success rates follow: $Geomob \approx GreedyFlow > SimBet \approx PER$. Such a result is consistent with those in Figures 7(a), 8(a), and 9(a) for the same reasons. We also find that when the memory size on each node increases, the success rates of all methods increase. This is because when the memory size increases, each node can carry more packets. This means that the capacity of the network is enhanced, leading to more successful packets.

4.3.2. Average Delay

Figures 10(b), 11(b), and 9(d) plot the average delays of the four methods in the experiments with different memory sizes using the DART, DNET, and ROMA trace, respectively. We find from the three figures that the average delays follow: $Geomob < GreedyFlow < SimBet < PER$. We see that this relationship is the same as in Figures 7(b), 8(b), and 9(b) due to the same reasons. Similarly, when the memory size on each node increases, the average delay in all methods decreases. This is because when the memory size increases, more packets can be carried by nodes that are most likely to deliver them to their destinations, thereby reducing the average delay of successfully delivered packets.

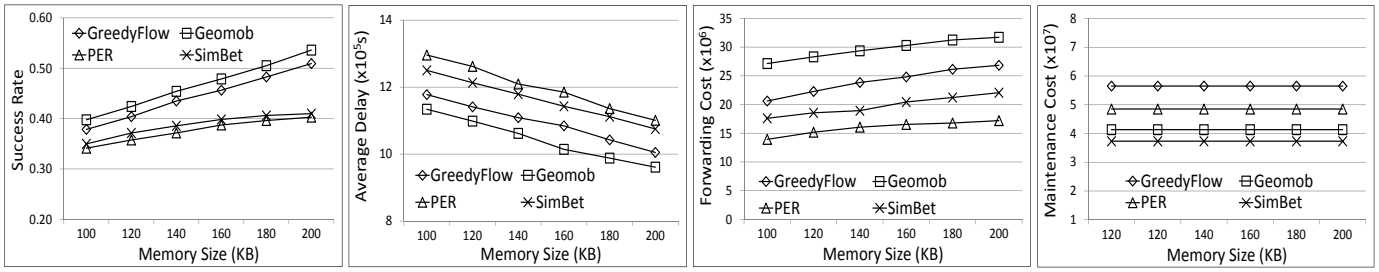
4.3.3. Forwarding Cost

Figures 10(c) and 11(c) show the forwarding costs of the four methods in the experiments with different memory sizes using the DART trace and the DNET trace, respectively. The two figures show that the average forwarding costs follow: $Geomob > GreedyFlow > SimBet > PER$. Again, this is the same as in Figures 7(c) and 8(c) due to the same reasons. We also find that when the memory size increases, the forwarding costs of all methods increase. This is because when each node can carry more packets, there are more packet forwarding.

4.3.4. Maintenance Cost

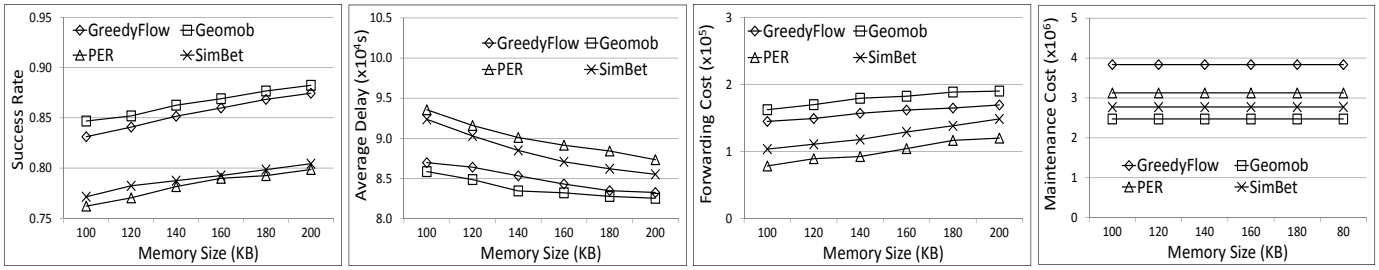
Figures 10(d) and 11(d) show the maitenance costs of the four methods in the experiments with different memory sizes using the DART trace and the DNET trace, respectively. The two figures show that the maintenance costs follow: $GreedyFlow > SimBet > PER > Geomob$

We can find that this relationship is the same as in Figures 7(d) and 8(d). Also, the maintenance costs remain unchanged with different packet rates or memory sizes. This is because the maintenance costs of these methods are only affected by the number of node encountering. Since we use the same traces in the each test, the maintenance costs remain the same. The results with different memory sizes further confirm the superior performance of GreedyFlow.



(a) Success rate. (b) Average delay. (c) Forwarding cost. (d) Maintenance cost.

Figure 10: Performance with different memory sizes using the DART trace.



(a) Success rate. (b) Average delay. (c) Forwarding cost. (d) Maintenance cost.

Figure 11: Performance with different memory sizes using the DNET trace.

4.4. Evaluation of Advanced Schemes

In this section, we evaluate the two advanced components for better local traffic map consistency (Section 3.4.4) and higher routing efficiency (Section 3.7). We denote the GreedyFlow with the two components as GreedyFlow-AdvC and GreedyFlow-AdvR, respectively.

4.4.1. Settings

We followed the settings in Section 4.1 to compare GreedyFlow, GreedyFlow-AdvC, and GreedyFlow-AdvR under different packet rates and different memory sizes. Since the experiments with the three traces show a similar trend (as shown in previous experiments), we only show the results with the DART trace in this section.

4.4.2. Different Packet Rates

We first test the two advanced components' performance with different packet rates. The results are shown in Figures 12(a), 12(b), 12(c), and 12(d). We see from those figures that the two advanced schemes lead to better routing performance. On average of all data rates, GreedyFlow-AdvC and GreedyFlow-AdvR improves the success rate by 6.1% and 9.7%, respectively, and reduces the average delay by 19000s and 26000s, respectively.

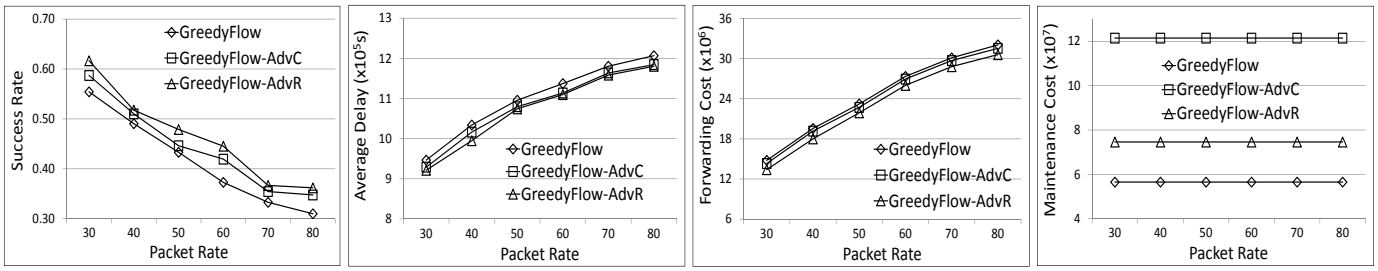
GreedyFlow-AdvC improves the routing performance by offering a better consistency among local traffic maps, so that packets can be more steadily forwarded towards destinations. GreedyFlow-AdvR leads to better routing performance by further exploiting individual nodes' preferences on visiting certain landmarks. As a result, packets can be forwarded to their destinations more quickly when they are close to their destinations. Since GreedyFlow-AdvR improves the routing performance more directly, it generates higher success rate and lower average delay than GreedyFlow-AdvC.

The two advanced components reduce the forwarding cost (i.e., number of forwarding operations) by 449360 and 1395182 on average, respectively, as shown in Figure 12(c). In GreedyFlow-AdvC, better consistency among local traffic maps reduces the chance of detour in the forwarding path. In GreedyFlow-AdvR, when a packet is forwarded to a node that can frequently visit its destination landmark, it will be carried by the node without being further forwarded. Thus, the forwarding cost is reduced.

The benefits of the two advanced components come at the cost of higher maintenance cost, which increases by 114% and 32%, respectively, as illustrated in Figure 12(d). In GreedyFlow-AdvC, encountered nodes exchange not only their transit frequencies among landmarks in local traffic maps but also accumulated transit frequencies of other nodes. As a result, it doubles the maintenance cost. In GreedyFlow-AdvR, two encountered nodes further exchange their visiting frequencies to top frequently visited landmarks. The amount of such information can be controlled by limiting the number of frequently visited landmarks. Therefore, GreedyFlow-AdvR only leads to moderate maintenance cost increase.

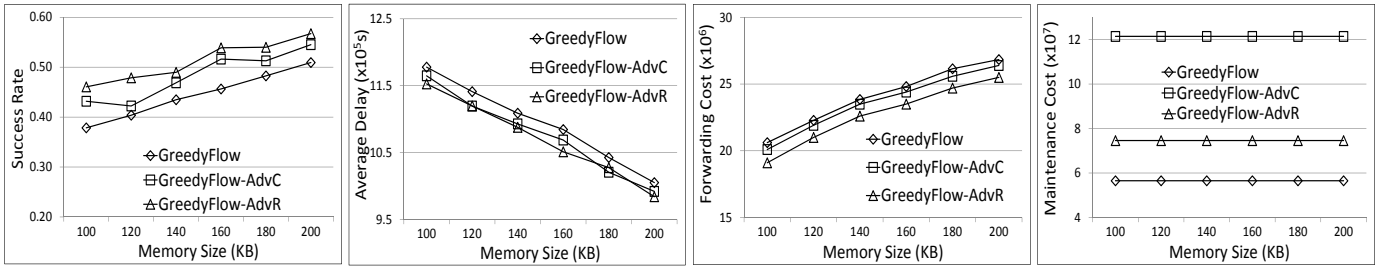
4.4.3. Different Memory Sizes

We further evaluate the performance of the two advanced components with different memory sizes. The results are shown in Figures 13(a), 13(b), 13(c), and 13(d). We find that the performance relationship between GreedyFlow,



(a) Success rate. (b) Average delay. (c) Forwarding cost. (d) Maintenance cost.

Figure 12: Evaluation of advanced components with different packet rates.



(a) Success rate. (b) Average delay. (c) Forwarding cost. (d) Maintenance cost.

Figure 13: Evaluation of advanced components with different memory sizes.

GreedyFlow-AdvC, and GreedyFlow-AdvR follows the same trend as in the test with different packet rates. On average of all memory sizes, the two components increases the success rate by 8% and 15%, decreases the average delay by 17000s and 22000s, reduces the forwarding cost by 44000 and 135000, increases the maintenance cost by 114% and 32%, respectively. Such a result further demonstrates that both advanced components can enhance the routing efficiency at the cost of more maintenance cost.

Combine all above experiment results, we think the two advanced components further improve the performance of GreedyFlow by improving the consistency of local traffic maps and by taking advantage of the node-based forwarding. However, the benefits come at more maintenance cost. Therefore, they can be implemented as optional features that can be enabled by the network operator based on actual needs. For example, the application that utilizes vehicles to transfer data in the mountain area would require a high routing efficiency and can tolerate more maintenance cost, as resources on vehicles often are abundant.

5. Conclusion

Data transmission between different places (landmarks) in a DTN can be used in many applications. In this paper, we propose a novel algorithm, namely GreedyFlow, to route packets between landmarks in DTNs in a fully distributed manner. To better utilize node mobility, GreedyFlow forwards packets in a landmark-by-landmark manner to let them gradually reach their destination landmarks. Each node collects node transit frequencies between landmarks in the area it primarily visits and uses such information to build a local traffic map. A global landmark that shows the distribution of landmarks is also built on each node off-line. The two maps are used to greedily forward packets toward their destination landmarks. The routing efficiency is further improved by two advanced components that enhance the consistency of distributively maintained local traffic maps and exploit node-based forwarding to reduce the expected delay, respectively. Extensive real trace driven experiments show that GreedyFlow has better performance than state-of-art routing algorithms and can achieve performance comparable to the routing algorithm that requires the global traffic information on each node. In the future, we plan to further enhance routing efficiency by considering social communities in DTNs.

Acknowledgements

This research was supported in part by U.S. NSF grants OAC-1724845, ACI-1719397 and CNS-1733596, Microsoft Research Faculty Fellowship 8300751, IBM Ph.D. fellowship award 2017 and the startup fund of SIU. A short version of this paper has been published in the Proc. of MASS'15 [35].

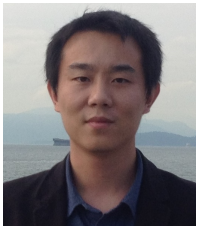
References

[1] S. Jain, K. R. Fall, R. K. Patra, Routing in a delay tolerant network, in: Proc. of SIGCOMM, 2004.

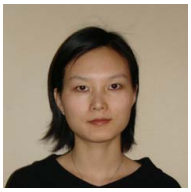
[2] K. Fall, A delay-tolerant network architecture for challenged internets, in: Proc. of SIGCOMM, 2003.

[3] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, D. Rubenstein, Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet, in: Proc. of ASPLOS-X, 2002.

[4] U. Weinsberg, A. Balachandran, N. Taft, G. Iannaccone, V. Sekar, S. Seshan, CARE: Content aware redundancy elimination for disaster communications on damaged networks, CoRR.
[5] J. Kurhinen, J. Janatuinen, Geographical routing for delay tolerant encounter networks, in: Proc. of ISCC, 2007.
[6] I. Leontiadis, C. Mascolo, GeOpps: Geographical opportunistic routing for vehicular networks, in: Proc. of WOWMOM, 2007.
[7] J. Lebrun, C. nee Chuah, D. Ghosal, M. Zhang, Knowledge-based opportunistic forwarding in vehicular wireless ad hoc networks, in: Proc. of VTC, 2005.
[8] Y. Cao, Z. Sun, N. Wang, M. Riaz, H. Cruickshank, X. Liu, Geographic-based spray-and-relay (gsar): an efficient routing scheme for dtns, IEEE Transactions on Vehicular Technology 64 (4) (2015) 1548–1564.
[9] Y. Cao, K. Wei, G. Min, J. Weng, X. Yang, Z. Sun, A geographic multicopy routing scheme for dtns with heterogeneous mobility, IEEE Systems Journal 12 (1) (2018) 790–801.
[10] K. C. Lee, M. Le, J. H01rri, M. Gerla, LOUVRE: Landmark overlays for urban vehicular routing environments., in: Proc. of VTC Fall, 2008.
[11] K. Chen, H. Shen, DTN-FLOW: Inter-landmark data flow for high-throughput routing in DTNs., in: Proc. of IPDPS, 2013.
[12] L. Zhang, B. Yu, J. Pan, GeoMob: A mobility-aware geocast scheme in metropolitans via taxicabs and buses., in: Proc. of INFOCOM, 2014.
[13] B. Wu, H. Shen, K. Chen, Exploiting active sub-areas for multi-copy routing in vdtns, IEEE Transactions on Vehicular Technology.
[14] L. Yan, H. Shen, K. Chen, Mobit: Distributed and congestion-resilient trajectory-based routing for vehicular delay tolerant networks, IEEE/ACM Transactions on Networking.
[15] A. Lindgren, A. Doria, O. Schelén, Probabilistic routing in intermittently connected networks., Mobile Computing and Communications Review 7 (3).
[16] A. Balasubramanian, B. N. Levine, A. Venkataramani, DTN routing as a resource allocation problem., in: Proc. of SIGCOMM, 2007.
[17] K. Lee, Y. Yi, J. Jeong, H. Won, I. Rhee, S. Chong, Max-Contribution: On optimal resource allocation in delay tolerant networks., in: Proc. of INFOCOM, 2010.
[18] W. Gao, G. Cao, On exploiting transient contact patterns for data forwarding in delay tolerant networks., in: Proc. of ICNP, 2010.
[19] H. Chen, W. Lou, Contact expectation based routing for delay tolerant networks, Ad Hoc Networks 36 (2016) 244–257.
[20] F. Li, J. Wu, MOPS: Providing content-based service in disruption-tolerant networks, in: Proc. of ICDCS, 2009.
[21] P. Hui, J. Crowcroft, E. Yoneki, Bubble rap: social-based forwarding in delay tolerant networks, in: Proc. of MobiHoc, 2008.
[22] E. M. Daly, M. Haahr, Social network analysis for routing in disconnected delay-tolerant MANETs, in: Proc. of MobiHoc, 2007.
[23] J. Wu, M. Xiao, L. Huang, Homing spread: Community home-based multi-copy routing in mobile social network, in: Proc. of INFOCOM, 2013.
[24] X. Zhang, G. Cao, Transient community detection and its application to data forwarding in delay tolerant networks., in: Proc. of ICNP, 2013.
[25] B. Ravaei, M. Sabaei, H. Pedram, S. Valaee, Community-aware single-copy content forwarding in mobile social network, Wireless Networks (2017) 1–17.
[26] H. Zhou, V. C. Leung, C. Zhu, S. Xu, J. Fan, Predicting temporal social contact patterns for data forwarding in opportunistic mobile networks, IEEE Transactions on Vehicular Technology 66 (11) (2017) 10372–10383.
[27] J. Link, D. Schmitz, K. Wehrle, GeoDTN: Geographic routing in disruption tolerant networks, in: Proc. of GLOBECOM, 2011.
[28] A. Sidera, S. Toumpis, DTFR: A geographic routing protocol for wireless delay tolerant networks, in: Proc. of Med-Hoc-Net, 2011.
[29] T. Henderson, D. Kotz, I. Abyzov, The changing usage of a mature campus-wide wireless network, in: Proc. of MOBICOM, 2004.
[30] A. Balasubramanian, B. N. Levine, A. Venkataramani, Enhancing interactive web applications in hybrid networks, in: Proc. of MOBICOM, 2008.
[31] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, A. Rabuffi, CRAWDAD dataset roma/taxi (v. 2014-07-17), Downloaded from `https://crawdad.org/roma/taxi/20140717` (Jul. 2014). `doi:10.15783/C7QC7M`.
[32] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik.
[33] Q. Yuan, I. Cardei, J. Wu, Predict and relay: an efficient routing in disruption-tolerant networks., in: Proc. of MobiHoc, 2009.
[34] E. M. Daly, M. Haahr, Social network analysis for routing in disconnected delay-tolerant MANETs, in: MobiHoc, 2007.
[35] K. Chen, H. Shen, Greedyflow: Distributed greedy packet routing between landmarks in dtns, in: Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on, IEEE, 2015, pp. 199–207.



Kang Chen received the BS degree in Electronics and Information Engineering from Huazhong University of Science and Technology, China in 2005, the MS in Communication and Information Systems from the Graduate University of Chinese Academy of Sciences, China in 2008, and the Ph.D. in Computer Engineering from the Clemson University in 2014. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering at Southern Illinois University. His research interests include emerging wireless networks and software defined networking.



Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor in the Department of Computer Science at University of Virginia. Her research interests include distributed computer systems and computer networks, with an emphasis on content delivery networks, mobile computing, wireless sensor networks, cloud computing, big data and cyber-physical systems. She is a Microsoft Faculty Fellow of 2010, a senior member of the IEEE, and a member of the ACM.