DSearching: Using Floating Mobility Information for Distributed Node Searching in DTNs

Kang Chen, Haiying Shen, *Senior Member, IEEE*, and Li Yan

Abstract—In delay tolerant networks (DTNs), enabling a mobile node to search and find another interested mobile node is an important function in many applications. However, the movement of nodes in DTNs makes the problem formidable. Current node searching methods in disconnected networks mainly rely on fixed stations in the network and infrastructure-based communication to collect node position information, which is difficult to implement in DTNs. In this paper, we present DSearching, a distributed mobile node searching scheme for DTNs that requires no infrastructure. In DSearching, the entire DTN area is split into sub-areas, and each node summarizes its mobility information as both transient sub-area visiting record and long-term movement pattern. Upon arriving at a sub-area, a node generates a new visiting record for the sub-area and distributes it to nodes that are likely to stay in the previous sub-area, so that visiting records form a chain for the locators to trace the node. Each node also stores different parts of its long-term mobility pattern to long-staying nodes in different sub-areas for others to trace it when visiting records are absent. Considering that nodes in DTNs usually have limited resources, DSearching constrains the communication and storage cost in the information distribution while enabling efficient node searching. Advanced extensions that can further improve the searching efficiency is also proposed in this paper. Extensive trace-driven experiments with real traces demonstrate the high efficiency and high effectiveness of DSearching.

Index Terms—Node mobility, node searching, DTNs

1 Introduction

In recent years, delay tolerant networks (DTNs) [1] have attracted significant research interests. In DTNs, nodes are sparsely distributed, and no end-to-end connection can be ensured. Therefore, enabling a node to search and find another interested node is an important function for node management and many applications in DTNs. For example, in the DTN that exploits sensors on animals (e.g., ZebraNet [2]) for various purposes (e.g., environment monitoring and animal tracking), we may need to find a specific sensor for upgrade or repair. For a DTN in the battlefield, the system administrator needs to not only isolate a misbehaving device but also find the owner who carries the device. In the DTN formed by mobile device users, the node search function can help a person to find and meet another person distributively. Fig. 1 demonstrates the problem of node searching in DTNs, in which the *locator node* searches for the *target node*.

This paper addresses the node searching problem in DTNs embracing social network properties: nodes move with certain patterns and have skewed visiting preferences [3], [4], [5]. Such properties can be found in many scenarios. For example, in the DTN consisting of wild animals, animals usually move between different places for food, water, and flock gathering with certain patterns. For DTNs in a battlefield, vehicles and soldiers mainly move on specific routes. In the DTN consisting of mobile devices on a campus, device holders (i.e., students) often visit certain buildings repeatedly, e.g., library, department building, and dorm.

There are already some works [2], [6], [7], [8] for object tracking (i.e., animal and people) under the context of DTNs or mobile networks. In these methods, the position or mobility information of a target object is proactively collected and gathered to search for it. Specifically, these methods require either central stations [2], [7], [8] or infrastructure-based communication (i.e., GPSR [8] and satellite communication [6]) to collect node position information. Though these techniques are possible and even common in certain scenarios (e.g., campus), they are costly to use and even impractical in DTNs designed for certain areas, e.g., rural and mountain areas.

Therefore, decentralized node searching method is favorable in DTNs. One intuitive way for node searching is to follow the DTN routing algorithms [4], [9], [10], [11], [12], [13], [14]. These methods deduce a node's probabilities of meeting other nodes and always forward a message to the node that has higher probability of meeting its destination. Then, the node locator can find the target node by following the holders of the message destined to the target node. However, this scheme may lead to a long delay because it only provides indirect mobility information of the target node (i.e., other nodes' probabilities of meeting the target node). Without knowing the direct mobility information of the target node, a node locator cannot move actively toward the target node for efficient node searching.

In DTNs, nodes meet opportunistically with limited communication range, leading to frequent network partition and posing great challenges on the retrieval of node mobility information. Though the target node can be sensed by its

• *The authors are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29631. E-mail: {kangc, shenh, lyan}@clemson.edu.*

Manuscript received 18 Aug. 2014; revised 14 Dec. 2014; accepted 11 Feb. 2015. Date of publication 4 Mar. 2015; date of current version 1 Dec. 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TMC.2015.2409867

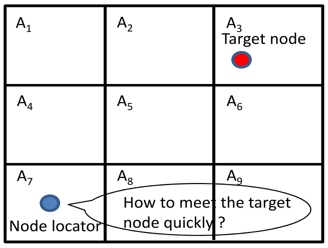


Fig. 1. Demonstration of node searching in DTNs.

neighbors, its position information can hardly be forwarded to the locator quickly. Further, the limited resources on mobile nodes make the broadcasting of mobility information or the storage of all mobility information on every node not applicable in DTNs. As a result, a distributed and lightweight algorithm is needed to distribute node mobility information in the network and make such information easily accessible for locators.

In this paper, we propose DSearching, a distributed and lightweight node searching algorithm in DTNs. The design of DSearching is based on two social network properties in DTNs. First, mobile nodes in DTNs usually exhibit certain movement patterns [5]. Indeed, one previous research reveals that the mobility of mobile devices carried by students on a campus is predictable [15]. Second, nodes always visit certain places and stay there for a relatively long time [4].

In DSearching, the entire DTN area is split into sub-areas, and each node collects its mobility information as both transient sub-area visiting records (VRs) and long-term movement pattern between sub-areas. Both types of mobility information are distributed to selected nodes to control the overhead and meanwhile support effective node searching. The transient sub-area visiting record indicates where the node just visits. Each node distributes its visiting record for a newly entered sub-area to nodes that are more likely to stay in the previous sub-area, so that the visiting records form a chain to enable locators to know the target's movement path. The long-term mobility pattern of a node in a sub-area reflects how it transits to next sub-areas in general. Each node distributes its long-term mobility pattern in a sub-area to long-staying nodes in the sub-area, and distributes its mobility pattern in its home-area to all sub-areas. Then, when visiting records are absent, such information can be used to search for the missing visiting records to recover the target node's movement path. In addition to above node mobility information, we also consider the correlation between sub-area visits as well as daily routine to further enhance node searching efficiency. Detailed analysis and evaluation demonstrate the effectiveness of DSearching.

In summary, our contributions are threefold:

1) We propose a lightweight method to distribute and store node mobility information on mobile nodes, which enables a locator to easily access such information.
2) We propose a node searching scheme that can efficiently and timely find a target in a decentralized manner.
3) We have conducted extensive trace-driven experiments to show the efficiency and effectiveness of DSearching.

The remainder of this paper is organized as follows. Related work is introduced in Section 2. Section 3 presents the detailed design of DSearching. In Section 4, the performance of DSearching is evaluated through trace-driven experiments. Section 5 concludes this paper with future work.

2 Related Work

Tracking objects in disconnected networks. Tracking objects in the network without constant connections has been studied in several previous works [2], [3], [6], [7], [8]. In SOMA [3], each node utilizes its previous records on encountering nodes and places to predict the places it is going to visit. ZebraNet [2] tracks Zebras in Kenya by configuring tracking collars on them, which record their positions and send the data back to the central station through hop-by-hop broadcasting. Cenwits [7] and SenSearch [8] aim to search people in wilderness areas without a connected network. They both utilize the opportunistic encountering among nodes to forward their position information to stations or access-points in the network. The work in [6] utilizes the flock behavior to reduce the cost needed to track sheep in wild areas. It basically lets the flock leader monitor and report the positions of other sheep to the server through GPRS or satellite communication. Different from these methods, DSearching does not require central stations, GPRS, or satellite communication to collect position information, but enables a node locator to actively find the target node in a completely distributed manner.

Routing in DTNs. Routing in DTNs has been widely studied in recent years [4], [9], [10], [11], [12], [13], [14]. These methods can generally be classified into two groups: probabilistic routing methods [9], [10], [11], [12] and geographical routing methods [4], [13], [14]. In the former group, RAPID [9] and MaxContribution [10] predict a node's future encountering probability with the destination based on previous encountering records and forward packets to nodes with higher probability of meeting their destinations. They also specify the forwarding or storage priorities of each packet based on their delivery probabilities. BUBBLE [11] and SimBet [12] further use social factors (i.e., centrality and similarity with the destination node) to deduce a node's probability of meeting a packet's destination.

In the latter group of methods, GeoDTN [13] predicts node encountering possibility based on previous movement and forwards a packet to nodes that are more likely to meet the destination node. GeoOpps [14] deduces each possible route's minimal estimated time of delivery (METD) to reach the closest point to the destination and forwards packets to the node with a smaller METD. LOOP [4] exploits mobility patterns of mobile nodes to predict their future movement to forward packets to certain areas in the network.

These algorithms only provide indirect information about the target node's position or mobility, i.e., how other nodes meet it. Therefore, they lead to a low node searching efficiency. On the contrary, DSearching uses a novel set of data structures to directly depict a node's mobility information. Therefore, DSearching enables a locator to actively move towards the target node, leading to more efficient node searching.

This paper is an extension of our conference paper [16]. Compared to the previous version, we have provided more insights on our design in this paper, i.e., the discussion on the design rationale and the efficiency of different components in our algorithm. We also have effectively extended the algorithm by exploiting more node mobility information, namely sub-area visit correlation and daily routine, to further improve the node searching efficiency.

3 System Design

We assume a DTN with n nodes denoted by N_i ($i = 1, 2, \dots n$). We regard the mobility of a node as the transits between sub-areas (different places) in the network. This raises a question on how to decide sub-areas? Clearly, the more sub-areas in the network, the more overhead is needed on maintaining node status, but also the more accurate depict of node mobility. Considering that the sub-area division is uniform for all nodes, we decide sub-areas based on popular places that are frequently visited by all nodes.

Specifically, we first select popular places in DTNs, which are common in DTNs with social network properties, e.g., villages in the DTNs on mountain or rural areas. Then, we partition the entire DTN area into sub-areas by below rules:

- Each sub-area contains only one popular place.
- The area between two popular places is evenly split to the two sub-areas containing the two places.
- There is no overlap among sub-areas.

Also, sub-areas with size smaller than a threshold (i.e., minimal sub-area size) are merged with the smallest neighbor sub-area. This is to prevent too many small sub-areas. Since nodes usually rely on short range communication techniques to communicate with each other in DTNs, we generally assume that nodes in one sub-area mainly only communicate with nodes in the same sub-area. Popular places and minimal sub-area size are decided by the network administrator.

The area partition is completed off-line to generate an area map with sub-areas. Each node is configured with the area map when it initially joins in the DTN with the DSearching enabled. This means that the map does not scale on individual node's visiting preference, and all nodes have the same map. Therefore, DSearching needs the application scenario information, i.e., popular places and general node mobility information, before applying it to a DTN. Though all sub-areas in the illustration in this paper have regular shapes, i.e., squares, they can be any shapes. Each sub-area is stored as the set of its vertexes in clock direction.

The design of the area map brings about extra overhead on 1) updating maps distributed to mobile nodes and 2) storage on mobile nodes. Fortunately, these extra overheads are not non-acceptable. For map update, we can generally assume that it will not be updated frequently, which means that the overhead in this step is controlled. For storage, the map only contains the ID and location of each sub-area, which do not occupy much storage space. Therefore, storing the map information won't be a burden for modern mobile devices.

We assume that each node has a GPS in DSearching. Thus, each node can know the sub-area in which it is located. The GPS is different from infrastructures since it can easily be installed on mobile devices. Furthermore, a node does not need to query the GPS frequently to save energy, i.e., it can query the GPS position only when finding that it may have transited to a new sub-area. Energy efficient localization techniques that use other signals such as WiFi can also be adopted to reduce the energy consumption for this purpose.

3.1 Representation of Node Mobility

In DSearching, each node records its mobility information for locators to search for it. First, to enable the locator to know its actual movement path, each node leaves transient sub-area visiting records as hints along the movement path. This is like the phenomenon in which an ant leaves a trail of pheromone for others to follow as it looks for food. Second, to enable the locator to find its regular movement pattern, each node creates a mobility pattern table (MPT) to summarize its staying in different sub-areas and its transits between sub-areas. We introduce the two types of mobility information below.

In this paper, we do not let nodes directly use their GPS positions to represent the mobility. There are two reasons for such a design. Firstly, it is hard to represent a node's mobility pattern through GPS positions. Nodes usually have the pattern about moving from one place to another place, but can hardly repetitively move from one GPS position to another GPS position. This means that we still need to map the GPS coordinates to places to generate mobility patterns. Secondly, since a small move would lead to a different GPS position, nodes need to frequently query the GPS module to update the GPS position, leading to a high energy consumption.

3.1.1 *Transient Visiting Record*

Each node leaves a hint, i.e., visiting record, when it enters a new sub-area:

$$VR :< N_i, A_{new}, GPS_{new}, A_{prev}, Time, T_s, Seq >,$$

where N_i is node ID, A_{new} and A_{prev} are the newly entered sub-area and the previous sub-area, respectively, GPS_{new} and $Time$ are the GPS position of the node and the time when the record is generated, respectively, T_s is the TTL of the VR, and Seq is the sequence number. Seq increases by 1 when a new VR is created. Considering that these records are time sensitive, we often set T_s to a relative short period of time, i.e., several hours or half day. A VR will be deleted once its TTL expires. The visiting record is designed to ensure that once the locator arrives at A_{prev}, it knows that the target node goes to A_{new} after moving out of A_{prev}.

3.1.2 *Long-Term Mobility Pattern*

A node's mobility pattern table contains two types of information for each of its regularly visited sub-areas: staying probability and transit probabilities. This is because nodes usually present skewed preferences on staying at certain sub-areas or transiting from a sub-area to another. The staying probability of a node, say N_i, at a

TABLE 1
Mobility Pattern Table on a Node

Rank	Sub-area	Staying Prob.	Next sub-areas and probabilities	Seq
1	A_5	0.50	$A_8(0.8)$, $A_2(0.2)$	1
2	A_2	0.25	$A_1(0.7)$, $A_3(0.3)$	1
3	A_6	0.15	$A_5(1)$	1
4	A_8	0.08	$A_7(0.6)$, $A_9(0.4)$	1
...	...	...	...	...

sub-area indicates how likely the node is in the sub-area and is calculated as

$$Ps_i(A_m) = D_m/D, \tag{1}$$

where D_m represents the total time N_i has stayed in sub-area A_m and D is the period of time the node has lived.

The transit probability of a node, say N_i, at a sub-area, say A_m, describes its probability to transit to another sub-area, say A_n, in the next movement, denoted $Pt_i(A_m \rightarrow A_n)$. It is calculated as below:

$$Pt_i(A_m \rightarrow A_n) = T_{mn}/T_{ma}, \tag{2}$$

where T_{mn} is the number of times that the node has transited from A_m to A_n and T_{ma} denotes the total number of times that the node moves out of sub-area A_m.

When a node's list of visited sub-areas becomes stable, it generates an initial MPT as shown in Table 1. Each node always maintains its own MPT. The MPT table will be updated when a new transit happens. After the update, the information about this transit can be discarded.

In the table, "Sub-area" records the regularly-visited sub-areas of the node, which are sorted in descending order of the staying probability. In the row for a sub-area, say A_m, the "Next sub-areas and probabilities" records the probabilities that the node moves from A_m to corresponding sub-areas. For example, $P_{ti}(A_2 \rightarrow A_1) = 0.7$, which means that the node's probability of transiting from A_2 to A_1 is 0.7. "Seq" is the sequence number of the table. It increases by 1 whenever the table is updated.

3.2 Mobility Information Distribution

We assume nodes are non-malicious and are willing to carry the mobility information of others to support the node searching function. We leave the work on how to motivate nodes to follow rules in DSearching to future work.

3.2.1 Distribute Visiting Record

When a node, say N_i, moves from sub-area A_m to sub-area A_n, it creates a visiting record as introduced in Section 3.1.1, denoted VR_{imn}. Recall that the purpose of the visiting record is to enable the locator to know the movement path of the target node. Following this direction, DSearching requires that each node to distribute VRs to nodes to ensure that the discovery probability that the locator can find them in the previous sub-area (e.g., A_m for VR_{imn}) is larger than a threshold Th_d. Therefore, Th_d is determined by general node density in sub-areas. The more nodes in a sub-area, the higher Th_d can be.

Specifically, we use $Pd_i(A_m)$ to denote the probability to find N_i's visiting records in A_m. Then, we copy VR_{imn} to nodes in A_n that is likely to transit to A_m and has a high probability to stay in A_m to satisfy that

$$Pd_i(A_m) = 1 - \prod_{r=1}^{k}(1 - Pt_r(A_n \rightarrow A_m) * Ps_r(A_m)) \geq Th_d, \tag{3}$$

where k is the number of selected nodes and $Pt_r(A_n \rightarrow A_m)$ and $Ps_r(A_m)$ denote the transit probability from A_n to A_m and the staying probability in A_m of the rth node, respectively.

We adopt two additional strategies to ensure that the above requirement is satisfied with a controllable amount of cost. Firstly, since each visiting record has a small TTL, we can select more nodes to hold VRs. Secondly, we can relay VRs to nodes that are more likely to stay in the previous sub-area. Therefore, we set Th_d to 0.8 in this paper.

Finally, whenever a node moves from one sub-area to another sub-area, visiting records are created to leave hint in the previous sub-area. As shown later in Section 3.3, these hints form a linked VR chain to help the locator find the target node along its actual path gradually and effectively.

3.2.2 Distribute the MPT

DSearching utilizes the storage on mobile nodes to store the MPT in a distributed manner for node searching. Since nodes usually have limited storage, it is not practical to store a node's MPT on every node. Also, a node's MPT should be easily accessed by its locators, which can appear in any sub-areas. Therefore, a node, say N_i, needs to choose a subset of nodes to store its MPT so that

- The locators for N_i can easily retrieve the MPT of N_i;
- The overhead for distributing N_i's MPT is controlled.

We take advantage of node mobility pattern and the mobility of locators to realize the two goals. The general method in DSearching is to only store the necessary part of a MPT in each sub-area for easy retrieval.

Storage host list. Recall that DSearching is proposed for DTNs with social network property that each node has several places it stays for a relatively long time [4]. This means that each sub-area tends to have nodes that often stay in it. We then define the nodes with staying probability in sub-area A_i larger than a pre-defined threshold as A_i's hosts, which is determined as the least staying probabilities of the top 30 percent mostly stayed nodes in A_i.

Then, in order to store a node's MPT in a sub-area, a certain number of hosts of the sub-area are selected to store the MPT. These nodes guarantee that even when nodes move continually, the probability that there is at least one copy of MPT stays in the sub-area is higher than a threshold (Th_t). In detail, each node maintains a *host list* for each sub-area containing the hosts in the sub-area that store its MPT, as shown in Table 2.

The "MPT staying prob." means the probability that at least one copy of MPT is in the sub-area and is calculated by $1 - \prod_{r=1}^{k}(1 - Ps_r(A_m))$, where k is the number of hosts and $Ps_r(A_m)$ is the rth host's staying probability in A_m.

TABLE 2
Host List for Each Sub-Area

Sub-area	Host list	Staying prob.	MPT staying prob.
Sub_1	N_1, N_2, N_4	0.90, 0.8, 0.7	0.994
Sub_2	N_9	0.99	0.99
...			
Sub_M	N_3, N_4	0.6, 0.7	0.88

Each node determines its host list during its movement. Specifically, suppose N_i tries to decide whether N_j can be added to its host list. For each sub-area in the table, N_j is temporarily added to the host list for the sub-area if: 1) N_j is a host of the sub-area with available memory, and 2) is not in the host list of the sub-area. Then, if the MPT staying probability of the sub-area is larger than Th_t, the host with the least staying probability is removed until the MPT staying probability reaches the minimum value that is no less than Th_t. Since Th_t is designed to ensure that the MPT staying probability in a sub-area is large enough, as Th_d, it is also determined by general node density in sub-areas. The host list of each sub-area can also be determined off-line based on node movement patterns (similar to the sub-area division process). We set $Th_t = 0.7$ in this paper.

MPT distribution and update. After creating the host lists from either on-line or off-line method, each node distributes its MPT to nodes in the host list for each sub-area when meeting them. In this step, DSearching does not store the entire MPT on each node. Instead, only a part of the MPT is stored on a node. Since the locator searches in a sub-area by sub-area manner, the MPT in each sub-area only needs to ensure that the locator knows where to search in the next step. For example, based on Table 1, a locator in A_5 only needs to know the row for A_5 in the target's MPT in order to know where to search in the next step.

Therefore, when N_i meets a node in its host lists, say N_j, N_i decides the content to be stored in N_j as below.

- N_i copies the first row of its MPT to N_j.
- If the home sub-area of N_j, denoted $hSub(N_j)$, exists in N_i's MPT, N_i copies the corresponding row to N_j.
- When a node receives the row of a MPT that it already has, it only keeps the latest one.

We define a node's home sub-area as the sub-area corresponding to the first row in its MPT, i.e., the sub-area that it stays for the most. This means that we can find a home sub-area for a node anyway, even when the node has similar staying probabilities in several sub-areas. The row for the home sub-area is distributed to all sub-areas because it is the sub-area that the node is most likely to stay in. Algorithm 1 summarizes the above MPT distribution process. Fig. 3 shows an example of the distribution of a MPT. We see that except the first row of the MPT, the hosts in sub-areas A_5, A_2, A_6 and A_8 only store their corresponding row in N_i's MPT.

We further analyzed two real traces to demonstrate that the effective home sub-area (i.e., the sub-area that a node visits clearly more often than others) commonly exists for most nodes. The two traces, namely DART and DNET, show the sub-area visits of mobile devices on a university campus and buses in a college town, respectively. The details of the two traces are introduced in Section 4.1. In the two traces, for each node, we calculated the ratio between its accumulated staying times in the top 2 mostly stayed sub-areas. We then show the CDF of such a ratio of all nodes in Fig. 2. We see from the figure that in both traces, for more than 80 percent of nodes, their staying time in the mostly stayed sub-area is 15 percent higher than that in the second mostly stayed one. This shows that nodes usually have an effective home sub-area, which further validates our design on MPT distribution.

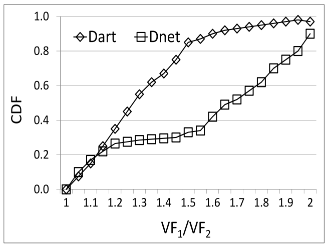


Fig. 2. The CDF of the ratio between the accumulated staying time in the mostly (VF_1) and the second mostly (VF_2) stayed sub-areas.

Algorithm 1. Pseudo-Code of the MPT Distribution when N_i Meets N_j.

```
1: procedure CheckToDistributeMPT($N_j$)
2:     if $N_j \in N_i$.HostList then
3:         $N_j \leftarrow$ the first row in $N_i$'s MPT
4:     else if $N_j$.HomeSubarea $\in N_i$.MPT then
5:         $N_j \leftarrow$ the row for $N_j$.HomeSubarea in $N_i$'s MPT
6:     end if
7: end procedure
```

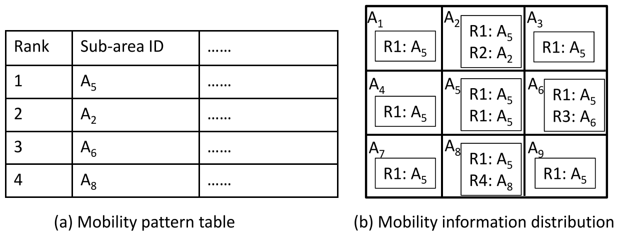
Finally, such a strategy can realize the aforementioned two goals. Firstly, wherever a locator starts searching for the target node, it can easily retrieve the needed mobility pattern information from hosts in current sub-area for efficient node searching. Secondly, each node only stores part of its MPT on a selected set of nodes, which saves the communication and storage cost on MPT distribution.

3.3 Node Searching

In node searching, we assume that the locator can move much faster than the target node, i.e., can pass more sub-areas in a unit time on average than the target node. This is reasonable because the locator is dedicated for node searching while the target node may sojourn at some places. When a locator moves slower than the target node, the searching efficiency will become very low in DSearching since it is mainly designed for locators to actively approach their targets. In that case, finding a place to wait for the target node would be more effectively, which is not the focus of this paper. We also assume that when the locator arrives at a sub-area, it can effectively search around to determine whether the target node is in the sub-area with a very high probability.

3.3.1 Overview

DSearching utilizes the visiting records and MPT rows distributed in the network to search for the target node. When a locator is initialized, it first searches the home sub-area of



Rank	Sub-area ID	......
1	A_5	......
2	A_2	......
3	A_6	......
4	A_8	......

(a) Mobility pattern table
A_1 R1: A_5
A_2 R1: A_5 R2: A_2
A_3 R1: A_5
A_4 R1: A_5
A_5 R1: A_5 R1: A_5
A_6 R1: A_5 R3: A_6
A_7 R1: A_5
A_8 R1: A_5 R4: A_8
A_9 R1: A_5
(b) Mobility information distribution

Fig. 3. Distribution of MPT entries (R1 denotes Row 1).

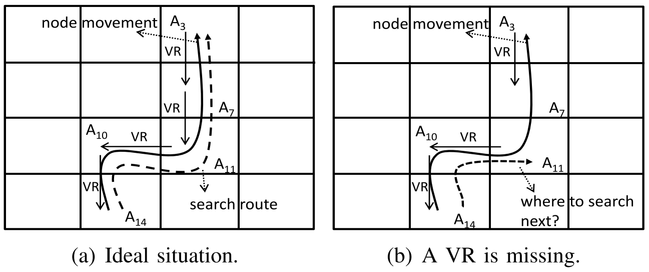


Fig. 4. Node searching with VRs.

the target. Then, considering the chain of visiting records provides information on actual node movement, the locator tries to follow the VR chain to search for the target node along its movement path. When a visiting record on the VR chain cannot be found, i.e., there is a gap on the VR chain, DSearching uses the MPT to retrieve a valid VR that can bridge the gap. During this process, whenever a valid visiting record is obtained, the locator moves to the A_{new} indicated in it. Above process repeats until the target is found.

3.3.2 Searching Startup

When a locator is initialized, it knows nothing about the mobility of the target node and can only search randomly. However, as mentioned in Section 3.2.2, the first row (home sub-area) of each node's MPT table is copied to all sub-areas. Therefore, the locator can easily know the target's home sub-area. Then, the locator moves to the home sub-area of the target to search for it. The rationale behind such a design is that the target stays longer in the home sub-area than in any other sub-areas.

3.3.3 Node Searching with VRs

Whenever the locator discovers one or more VRs of the target node that are newer than the current one it uses, it moves to the A_{new} in the latest VR, i.e., the one with the largest sequence number, to search for the target node based on the GPS_{new} in the record. In this sub-area, if the target node cannot be found, the locator is supposed to discover the VR indicating where the target node moves to from the sub-area. When the locator finds such a VR, it again goes to search the A_{new} in the VR. Ideally, following this manner, the locator searches for the target node along its movement path indicated in a chain of VRs. As shown in Fig. 4a, the locator searches along the actual movement path of the target with the help of VRs, i.e., $A_{14} \rightarrow A_{10} \rightarrow A_{11} \rightarrow A_7 \rightarrow A_3$.

3.3.4 Node Searching without VRs

Though the design in Section 3.2.1 requires that each VR should exist in the previous sub-area with a high probability ($\geq Th_d$), VRs actually are floating in the network due to node mobility. Therefore, it is common that a certain VR cannot be discovered by the locator, leading to a gap on the VR chain. As shown in Fig. 4b, the VR created in sub-area A_7 fails to reach A_{11}. Then, after searching A_{11}, the locator cannot know where to search for the next step. In this case, the MPT table and geographical limitations are jointly considered to provide an effective solution with a low cost.

Specifically, suppose the locator fails to find the expected VR in a sub-area, say A_{x_0}, that indicates where the target moves to from A_{x_0}. Then, the locator moves around to find the missing VR or a VR created after it. We define the cost in this process as the expected number of searching hops, denoted $\mathcal{W}$. One searching hop refers to the movement from one sub-area to another sub-area. Then, the goal is *minimizing the $\mathcal{W}$ needed to find a VR that can bridge the gap on the VR chain*. In below, we first assume that when arriving at a sub-area, the locator can obtain the corresponding row for the sub-area in the target's MPT from nearby nodes, which can easily be realized based on the design in Section 3.2.2. We then discuss the case when such MPT information cannot be obtained later.

Optimal solution. We first model the searching process to find the optimal searching route. Suppose there are total M sub-areas and the locator searches sub-areas in the sequence of $A_{x_1}, A_{x_2}, A_{x_3}, \ldots, A_{x_M}$. Then, the probability that a VR that can bridge the gap in the VR chain can be found in A_{x_r} ($r \in [1, M]$), denoted $Pf(A_{x_r})$, can be expressed as

$$Pf(A_{x_r}) = Pv(A_{x_r})Pd(A_{x_r}), \tag{4}$$

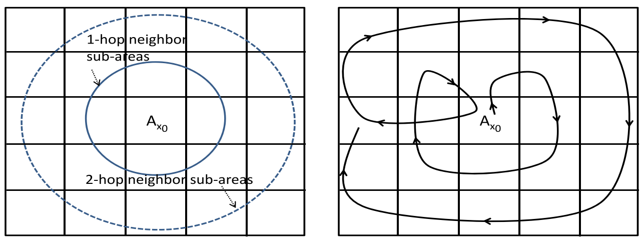
where $Pv(A_{x_r})$ denotes the probability that the target has visited A_{x_r} after moving out of A_{x_0}, and $Pd(A_{x_r})$ denotes the probability that the VR created in the sub-area immediately after A_{x_r} can be found in A_{x_r}, which is introduced in Equation (3). We then have

$$\mathcal{W} = \sum_{r=1}^{M} S(A_{x_{r-1}}, A_{x_r}) * \left(\prod_{s=1}^{r-1}(1 - Pf(A_{x_s}))\right) Pf(A_{x_r}), \tag{5}$$

where $S(A_{x_{r-1}}, A_{x_r})$ denotes the number of hops needed to move from $A_{x_{r-1}}$ to A_{x_r}. Therefore, the optimal searching route is a set of $A_{x_1^*}, A_{x_2^*}, A_{x_3^*}, \ldots, A_{x_M^*}$ that results in the minimal $\mathcal{W}$.

However, such a route can hardly be found. Firstly, $Pf(A_{x_r})$ cannot be obtained by the locator. For $Pv(A_{x_r})$, it should consider all possible moving paths from A_{x_0} to A_{x_r}. However, the locator can only know the target's transit probabilities in A_{x_0} from the MPT, which only shows the 1-hop path from A_{x_0} to neighboring sub-areas. For $Pd(A_{x_r})$, the information needed to calculate it (e.g., in Equation (3)) cannot be known by the locator. Secondly, $S(A_{x_{r-1}}, A_{x_r})$ depends on the locations of both $A_{x_{r-1}}$ and A_{x_r}. Therefore, even when $Pf(A_{x_r})$ is known, there is no efficient way to find the optimal searching route.

A practical method. In order to find a practical and effective method, we first investigate the geographical limitations. We define H-hop neighbor sub-area set of a sub-area, say A_{x_0}, as the set of sub-areas that a node needs at least H hops to reach them from A_{x_0}. In Fig. 5a, the 1-hop and 2-hop neighbor



(a) N-hop neighboring sub-areas. (b) A searching route.

Fig. 5. Node searching without VRs.

sub-areas of A_{x_0} are connected by two circles, respectively. We can easily find that after moving out of A_{x_0}, the target node needs to visit at least one H-hop neighbor ($H = 1$ or 2) sub-area in order to pass through the area covered by these sub-areas. Therefore, it is highly possible that a VR that can bridge the gap on the VR chain can be found in these sub-areas. We then limit the searching for VRs within 1-hop and 2-hop neighbor sub-areas. Considering the 1-hop sub-area set has smaller size than the 2-hop sub-area set (8 versus 16) and is closer to A_{x_0}, we let the locator first searches all 1-hop neighbor sub-areas and then all 2-hop neighbor areas. We do not let the locator repeatedly searching the 1-hop sub-areas if a valid VR cannot be found. This is because it is also highly possible to find a valid VR in the 2-hop sub-areas.

Furthermore, we let the locator only searches sequentially along the circle connecting all H-hop sub-areas, i.e., the circles in Fig. 5a. This is because the movement from one H-hop sub-area to another non-neighboring H-hop sub-area needs at least 2 and at most $2 * H$ hops, while the sequential searching takes only $8H$ hops to search all sub-areas. With this limitation, $S(A_{x_{r-1}}, A_{x_r}) = 1$ in Equation (7), which greatly reduces the complexity of finding the most efficient route.

With above simplification, when searching the H-hop neighbor sub-area set, we only need to determine the start sub-area and the searching direction, which has only $2 * 8 * H = 16H$ cases since we only have two possible directions and $8H$ start sub-areas. Specifically, for 1-hop neighbor sub-area set, Equation (4) can be calculated by

$$\begin{aligned} Pf(A_{x_r}) &= Pv(A_{x_r})Pd(A_{x_r}) \\ &= Pt(A_{x_0} \rightarrow A_{x_r}) * Th_d, \end{aligned} \tag{6}$$

where $Pv(A_{x_r})$ and $Pd(A_{x_r})$ are approximated by $Pt(A_{x_0} \rightarrow A_{x_r})$ and Th_d, respectively. For the former, the simplification only considers the 1-hop transit from A_{x_0} to A_{x_r}, i.e., $Pt(A_{x_0} \rightarrow A_{x_r})$, because 1) the 1-hop transit accounts for the majority of all transits and 2) this is the only information that the locator can get from the target's MPT so far. For the latter, since DSearching always try to ensure that $Pd(A_{x_r})$ is larger than Th_d, which is set to a large number, e.g., 0.8, it is acceptable to use Th_d to represent $Pd(A_{x_r})$. Then, DSearching calculate the $\mathcal{W}$ for all 16 cases, i.e., 8 possible start sub-areas and each has two directions, by below

$$\begin{aligned} \mathcal{W} &= \sum_{r=1}^{M} S(A_{x_{r-1}}, A_{x_r}) * \left(\prod_{s=1}^{r-1} (1 - Pf(A_{x_s})) \right) Pf(A_{x_r}) \\ &= \left(\prod_{s=1}^{r-1} (1 - Pf(A_{x_s})) \right) Pf(A_{x_r}). \end{aligned} \tag{7}$$

Then, the start sub-area and searching direction that lead to the minimal $\mathcal{W}$ is selected.

After searching all 1-hop neighbor sub-areas, if the VR that can bridge the gap on the VR chain is not found, the locator then searches the 2-hop neighbor sub-area set. This process is the same as that for the 1-hop neighbor sub-areas except for the calculation of $Pf(A_{x_r})$. In this case,

$$Pv(A_{x_r}) = \sum_{r=1}^{R_r} Pt(A_{x_0} \rightarrow A_{y_r}) * Pt(A_{y_r} \rightarrow A_{x_r}), \tag{8}$$

where A_{y_r} denotes the intermediate 1-hop sub-area through which the target node can move from A_{x_0} to A_{x_r}, and R_r is the number of such sub-areas. Then, the start sub-area and searching direction that lead to the minimal $\mathcal{W}$ is selected to search all 2-hop neighbor sub-areas. Note that $Pt(A_{y_r} \rightarrow A_{x_r})$ can be learned from the MPT rows for 1-hop neighbor sub-areas, which are obtained in the previous step.

Fig. 5b demonstrates the searching of the 1-hop and 2-hop neighbor sub-areas. During this process, if a VR that can bridge the gap on VR chain can be found, the locator simply follows the VR to continue the search. If not, the locator moves randomly out of the areas covered by the 1-hop and 2-hop neighbor sub-areas to search for VRs.

Without MPT information. Due to node mobility, it is possible that the MPT information in a 1-hop or 2-hop neighbor sub-area cannot be obtained, which means that the locator cannot know corresponding $Pt(A_{x_0} \rightarrow A_{x_r})$. In this case, we simply regard that each $Pt(A_{x_0} \rightarrow A_{x_r})$ equals to the average value, i.e., all transits have the same possibility.

3.4 Summary of the Behaviors of Nodes and Locators

We summarize the behaviors of nodes and locators in DSearching. For mobile nodes, they first collect enough movement records to create the mobility pattern table as introduced in Section 3.1.2. Meanwhile, when a node enters a new sub-area, it creates a visiting record as introduced in Section 3.1.1. Both visiting records and mobility pattern tables are distributed to nodes in the network following the methods in Sections 3.2.1 and 3.2.2, respectively.

The locator first moves to the home sub-area of the target, which can be known from the collected MPT of the target node, to search for the target, as introduced in Section 3.3.2. Then, from the home sub-area, the locator follows the VR chain to search for the target along its actual movement path, as introduced in Section 3.3.3. In case an expected VR cannot be found, i.e., there is a gap on the VR chain, the locator follows the method in Section 3.3.4 to find such a VR. Once an expected VR is found, the locator again moves along the VR chain to search for the target node. Such a process repeats until the target node is found.

3.5 Advanced Extensions of DSearching

We further introduce additional mobility information that can be used to enhance the node searching efficiency. First, in DTNs with social network properties, the sub-areas that a mobile node frequently visits often are not independent. For example, in DTNs on campus, students are likely to go to the library after finishing lectures in their department buildings; professors often go back to their offices after giving lectures.

We denote such a correlation as sub-area visiting correlation. Second, the transit pattern in each day tends to be stable for many nodes. For example, a Ph.D. student often attends classes from 9AM to 10AM and 2PM to 3PM each Monday. We define such a pattern as the daily routine.

The locator can better predict a node's staying in different sub-areas by considering its sub-area visiting correlation and daily routine. However, when related extensions are not enabled or the necessary information (e.g., matched routine information) is not available, locators simply follow the basic scheme to search for the target node. We introduce how to utilize them for more efficient node searching below.

3.5.1 MPT Considering Sub-Area Visiting Correlation

In the mobility pattern table described in Section 3.1.2, a node's transit probability from one sub-area to another sub-area is solely determined based on the frequency of such 1-hop transits, as shown in Equation (2). In other words, it assumes that a node's next transit is independent with its previous transits. However, in some cases, such an assumption may not be accurate and may lead to misleading results.

We take the transit records of Node 27 in the Dartmouth trace (DART) [17] as an example to illustrate this point. Through analyzing the trace, we find that the frequency of transit "$AcadBldg34 \rightarrow ResBldg82$" is larger than that of the transit "$AcadBldg34 \rightarrow LibBldg4$". Thus, according to the design of current MPT, the probability that after staying in $AcadBldg34$, the node will move to $ResBldg82$ is larger than the probability that the node will move to $LibBldg4$. Nevertheless, if we consider the previous transit through which the node moves to $AcadBldg34$, the next hop transit probability would be different. Specifically, the frequency of transit "$ResBldg82 \rightarrow AcadBldg34 \rightarrow LibBldg4$" is larger than that of the transit "$ResBldg82 \rightarrow AcadBldg34 \rightarrow ResBldg82$". This means that if node 27 transits to $LibBldg4$ from sub-area $ResBldg82$, it is more likely to transit to $LibBldg4$ after staying at sub-area $AcadBldg34$. Such a case shows that the previous transit of a node tends to have a certain correlation with its next transit, which can be utilized to enhance the accuracy of node transit prediction. Below, we first introduce the improved MPT based on such conditional transit probabilities and then present how to utilize the improved MPT for more efficient node searching.

Improved MPT. With above findings, we try to integrate previous sub-areas into the MPT to more accurately deduce the probabilities on which sub-area the node may move to in the next transit. Based on our previous investigation [18], considering previous one hop transition can lead to the highest accuracy on the prediction of the next hop transit. Therefore, we mainly consider the transit probabilities in condition of the previous transit in this paper. In detail, we partition the overall transit records into two categories: the transit without previous sub-area, which is referred to as *static transit*, and the transit with previous sub-area, which is referred to as *dynamic transit*. We then design an improved mobility pattern table that contains the transit probabilities for both static transit and dynamic transit. The probability of a static transit is calculated in the same way as in Equation (2). The probability of a dynamic transit is calculated on condition of the previous transit. For instance, the transit record of a node, say N_i, at a sub-area, say A_m, describes its intention to transit to another sub-area, say A_n, in next movement. The last transit record of the node, say from sub-area A_l to A_m, describes where the node comes from. So the new transit probability can be denoted $P_{t_i}(A_m \rightarrow A_n | A_l \rightarrow A_m)$, which can be calculated as below:

$$P_{t_i}(A_m \rightarrow A_n | A_l \rightarrow A_m) = \frac{P(A_l \rightarrow A_m, A_m \rightarrow A_n)}{P(A_l \rightarrow A_m)} = \frac{T_{lmn}/(T_{N_i} - 1)}{T_{lm}/T_{N_i}} \approx \frac{T_{lmn}}{T_{lm}}, \quad (9)$$

where $P(A_m \rightarrow A_n, A_l \rightarrow A_m)$ is the node's probability of transiting from sub-area A_m to sub-area A_n after moving from sub-area A_l to sub-area A_m. It is calculated as the number of the two-hop transits, denoted T_{lmn}, over the total number of transits of the node minus 1, denoted $T_{N_i} - 1$; $P(A_l \rightarrow A_m)$ is the probability of the node moving from sub-area A_l to sub-area A_m, which is represented as the number of the transits from sub-area A_l to sub-area A_m, denoted T_{lm}, over T_{N_i}.

TABLE 3
Improved Mobility Pattern Table with Static and Dynamic Transits

Rank	Present sub-area	Staying prob.	Previous sub-area	Next sub-areas and prob.
1	A_5	0.5	A_3	$A_8(0.74)$, $A_4(0.13)$, $A_2(0.13)$
			A_1	$A_8(0.4)$, $A_6(0.6)$
			-	$A_8(0.6)$, $A_6(0.2)$, $A_4(0.1)$, $A_2(0.1)$
4	A_2	0.25	A_9	$A_1(1.0)$
			A_4	$A_3(1.0)$
			A_5	$A_1(0.6)$, $A_3(0.4)$
			-	$A_1(0.7)$, $A_3(0.3)$
3	A_6	0.15	-	$A_5(1.0)$

Table 3 presents one example of the improved mobility pattern table. In the table, for each present sub-area, the sub-row that takes "-" as the previous sub-area represents the probabilities for static transits, and each sub-row with a specific sub-area as the previous sub-area denotes the probabilities for dynamic transits. Further, the staying probability in a sub-area is calculated following the same method in the original MPT, i.e., based on the portion of staying time in the sub-area.

Utilizing the improved MPT. The improved MPT is distributed following the same way for the original MPT, as introduced in Section 3.2. We then present how to use the improved MPT to further improve the searching efficiency. Specifically, when tracing a node to a sub-area, the locator will consider the previous sub-area that the node transits to this sub-area to predict the next sub-area that the node transits to, i.e., the sub-area to search in the next step. For example, suppose Table 3 is the improved MPT for a target node, if the locator for the target node arrives at A_5 and finds that the target node previously moves to A_5 from A_1. Then, following the dynamic transit probabilities, the locator searches around to find the missing VR as introduced in Section 3.3.4.

TABLE 4
Routine Table

Node 0			
Entry	Time interval	Landmark ID	Prob.
1	$08:00 \sim 09:00$	$AcadBldg34$	0.43
2	$10:00 \sim 12:00$	$AcadBldg8$	0.21
...	...	...	...

TABLE 5
Transient Routine Record

Node ID			
Seq	0	TTL	30 min
Entry	Interval of time	Landmark ID	Prob.
1	$08:00 \sim 09:00$	$AcadBldg34$	0.43
2	$10:00 \sim 12:00$	$AcadBldg8$	0.21

3.5.2 Exploiting the Daily Routine

In this subsection, we explain how to represent and distribute nodes' daily routine information, as well as the utilization of such information for more efficient node searching.

Motivation. Through the analysis of the Dartmouth trace [17] and DieselNet AP trace [19], we found that a large part of nodes have preferences in visiting sub-areas. Moreover, the intervals of time during which nodes visit their preferreed sub-areas are relatively more stable than other rarely visited sub-areas. Take the transit preference of Node 0 as an example, the most frequently appeared transit record is from $AcadBldg34$ to $AcadBldg8$. Also, quite naturally, $Node\ 0$ often moves from $AcadBldg34$ to $AcadBldg8$ within the time interval between 1:30PM and 2:30PM. This matches with our daily experience that people usually have certain routine on visiting places such as work place, supermarket, and home.

Such routine information can help realize more efficient node searching. After discovering the daily routine of the target node, the locator can compare such information with the current system time and then decide whether to go to the places indicated in the routine information to search for the target node directly. With such an improvement, the locator can quickly move to places where the target node visits regularly, thereby reducing the searching delay.

Summarizing routine information. We build a routine table to summarize a node's routine information. The routine table has three types of records: $Time\ interval$, $Landmark\ ID$, and $Probability$. The $Time\ interval$ indicates during what periods of time the node has routines in visiting places. The $Landmark\ ID$ indicates where the node will possibly be in the corresponding interval of time. The $Probability$ describes the probability of staying in $Landmark\ ID$ during the $Time\ interval$. Only the time intervals with probability larger than a threshold, e.g., 0.2, have a corresponding entry in the routine table. This is because only stable routines can help node searching. The detailed structure of the routine table is shown in Table 4. Note that the time interval in the routine table is not fixed since a node's stays in different landmarks vary case by case.

The key issue with the routine table is how to determine the list of time intervals. This is because the lengths of a node's stays in differents landmark may vary significantly. This means that the time intervals for the routinely visited landmarks tend to be not stable. As a tradeoff, the mostly overlapped time interval for a landmark will be chosen as the time interval for the routine table. Then, the probability for the ith time interval is calculated as

$$P_{r_i}\{Node\ 0\ is\ in\ LM_k\ during\ Interval\ i\} = \frac{T_{i,LM_k}}{T}, \quad (10)$$

where the T_{i,LM_k} is the accumulated amount of time that Node 0 stays in LM_k during interval i and T is the total active time of Node 0 during interval i.

Distribution of routine table. Duplicating the whole routine table is exhausting and unnecessary. To control the storage overhead, the size of the distributed routine table should be limited. For a locator, the interval of time covering present system time is much more important than other intervals listed in the routine table. This is because the real-time location of the target is the information that the locator cares for the most. Thus, we partition the routine table into several small-sized units, which are called transient routine records. Each transient routine record consists of two entries of the routine table, which are the entry covering current system time and the entry covering the next time interval. The structure of a transient routine record is shown in Table 5, in which the current system time is 8:30AM.

To disseminate the routine information of each node efficiently, every node is required to generate a transient routine record from its routine table periodically, e.g., every 1 hour, according to the system time. Then, each node only distributes the current transient routine record to nodes it meets until the generation of the next record. Thus the routine information of each node is floating in the network.

On the other hand, in order to provide reference for the locator, each node maintains a first-in-first-out queue to store the transient routine records it receives from other nodes. All records in the queue have identical time to live (*TTL*) that decreases 1 per time unit. The queue is updated only under four cases: 1) The new record comes from the node that the queue doesn't cover. In this case, the new record will be allowed to enter the queue. 2) The queue already has a record of the node that the new record belongs to but the new record covers a different interval of time. The new record will also be allowed to enter the queue in this case. 3) The *TTL* of some records has reduced to 0, the record will be removed from the queue. 4) If the new record is a newer version of an existent record in the queue, the new version of the record will replace the old version one in the queue. However, in this case, the *TTL* of the record is still the old version one, i.e., not refreshed. As a result, the routine table of each node will be disseminated and updated constantly in the network.

We set the maximum queue length to be the number of nodes subtracting one. This is because the routine table of the node holding the queue is not needed to be stored in the queue. When the number of entries in the queue is less than the upper bound, the arrived transient routine record that is allowed to enter the queue will be pushed into the queue directly. If the queue is full, an entry of the node having the

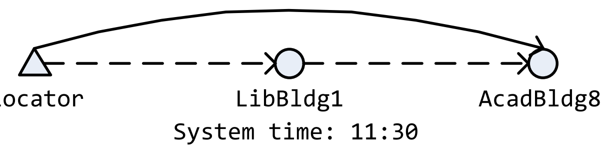


Fig. 6. Demonstration of routine table aided node searching.

most transient routine records will be removed. Such a setting is to ensure the fairness among nodes in the queue.

Routine table aided node searching. After running for some time, every node can generate a summary of its daily routines in the form of routine table based on its historical transit records. Each node also distributes transient routine records to other nodes according to the method mentioned in the previous subsection. The routine table can be used as an auxiliary positioning method for node searching.

Specifically, each time the locator meets a node, it tries to fetch the transient routine record of the target node from the newly encountered node. Then, the locator can find the most possible landmark(s) where the target will be based on current system time and the collected routine information. The locator can directly move to the indicated landmark(s) to search for the target. Fig. 6 shows one case that the search speed is accelerated through utilizing the routine information. Suppose the target node, Node 0, currently is at sub-area $AcadBldg8$, and the system time is 11:30AM. We also assume a received transient routine record is as shown in Table 5. Generally, to reach sub-area $AcadBldg8$, the locator has to go through sub-area $LibBldg1$. Now with the second entry in Table 5, the locator knows that Node 0 is most likely to stay in sub-area $AcadBldg8$ now. Thus, rather than searching $LibBldg1$ first, it moves towards sub-area $AcadBldg8$ along the geographically closest path directly.

On the other side, though the routine information is useful in indicating the places that the target node is likely to be, VRs and MPT information are the primarily way to decide the searching direction. This is because the routine table may only cover a short period of time in each day, i.e., a node may have a routine only in a part of time daily. Therefore, the routine table will be used to infer the possible position of the target node only when available.

4 Performance Evaluation

In this section, we evaluate the performance of DSearching with two real DTN traces and the trace obtained by nine students carrying a mobile phone on our campus. We first disable the advanced extensions mentioned in Section 3.5 to show the advantage of basic DSearching. We then evaluate the effectiveness of the advanced extensions in Section 4.8.

4.1 Empirical Datasets

The first trace, denoted Dartmouth trace [17], records the association of students' digital devices with WiFi APs on Dartmouth campus. We regarded each building as a sub-area and merged neighboring records for the same mobile device and the same sub-area. We also removed short connections (<200 s) and nodes with few records (<500). Finally, we obtained 320 nodes and 159 sub-areas.

The second trace, namely DieselNet AP trace (DNET) [19], collects the WiFi AP association records of 34 buses in the downtown area of a college town. Since there are many APs that do not belong to the experiment in the outdoor environment, APs with few appearances (<50) were removed from the trace. We mapped APs that are within certain distance ($<1.5\,\mathrm{km}$) into one sub-area. The trace is pre-processed similarly as that for the DART trace. Finally, we obtained 34 nodes and 18 sub-areas.

TABLE 6
Characteristics of Mobility Traces

	DART	DNET
# Nodes	320	34
# Sub-areas	159	18
Duration	119 days	20 days
# Transits	4,77,803	25,193

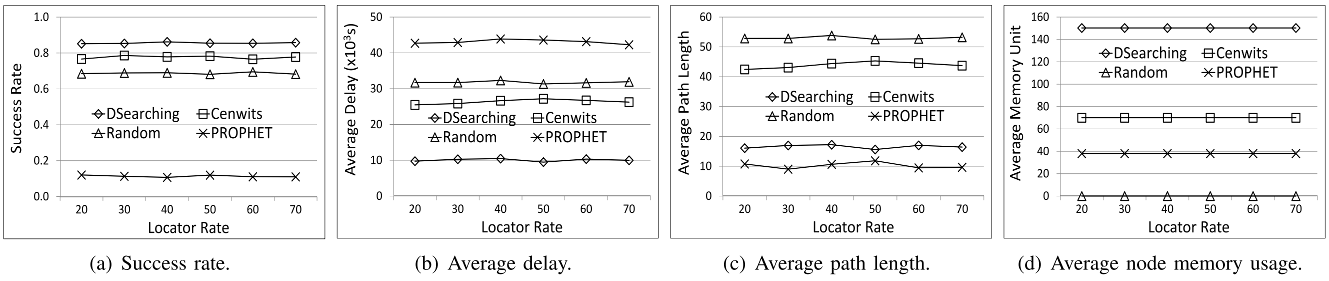
The characteristics of the two traces are shown in Table 6. In the table, "# Transits" refers to the total number of transits between sub-areas performed by all nodes.

4.2 Experiment Setup

Considering the length of the two traces, we set the initial period to 30 days for the DART trace and 2.5 days for the DNET trace, during which nodes collect mobility information to build the MPT. Then, locators were generated with random start sub-area and target node at the rate of R_p per day, which was set to 40 by default. Considering students move less frequently than buses, the default locator TTL was set to 24 hours in the DART trace and 4 hours in the DNET trace. Since both traces do not provide the map information, we assume that the locator needs 10 minutes to move from one sub-area to another sub-area on average.

We compared DSearching with three representative methods: an encountering based method (Cenwits) [7], a routing based method (PROPHET) [20], and a random searching method (Random). In Cenwits, nodes record their meeting locations and times with other nodes and exchange such information with others. The locator collects such information from encountered nodes and moves to the most recent place where the target node appears to search for it. In PROPHET, the locator follows the node that has the highest possibility to meet the target node to search for it. In Random, the locator moves randomly to search for the target node. We pick the three comparison methods to show how different levels of node mobility information affect the node searching efficiency. Our method uses the disseminated location information and mobility pattern; Cenwits only uses disseminated location information; PROPHET uses indirect mobility information; Random uses no information at all.

We measured four metrics: *success rate*, *average delay*, *average path length*, and *average node memory usage*. The former three refer to the percentage of locators that successfully find their target nodes and the average delay and average path length of these locators, respectively. The last one denotes the average number of memory units used by each node. For DSearching, we take one row of the MPT and four visiting records as one memory unit. For Cenwits, 4 meeting records with others are regarded as one memory unit. For PROPHET, 8 meeting probabilities are regarded as one memory unit. We set the confidence interval to 95 percent in the paper.



(a) Success rate. (b) Average delay. (c) Average path length. (d) Average node memory usage.

Fig. 7. Performance with different locator rates using the DART trace.

4.3 Experiments with Different Locator Rates

In this test, we varied the locator rate R_p from 20 to 70.

4.3.1 Success Rate

Figs. 7a and 8a present the success rates of the four methods in the tests with the DART trace and the DNET trace, respectively. We see that the success rates follow: DSearching > Cenwits > Random > PROPHET.

PROPHET has the lowest success rate because a locator does not move actively to search for the target node but only follow the mobile node that has the highest probability to meet the target node, which has its own mobility pattern. Therefore, a lot of locators expire due to TTL, leading to the lowest success rate. For Random, locators move actively but search blindly, resulting in a low success rate. Cenwits explores the witness of the target node's appearances in different sub-areas to actively search for it. Therefore, it has higher success rate than Random and PROPHET. However, Cenwits has lower success rate than DSearching. This is because Cenwits only simply utilizes the recent appearances of the target node but neglects the mobility pattern of the target node. On the contrary, DSearching combines the two information to enable more efficient and accurate node searching.

We also find that except DSearching, other three methods have obvious higher success rate in the test with the DNET trace than in the test with the DART trace. This is because the DART trace represents a network with a lot of sub-areas (i.e., 159) while the DNET trace is a small scenario with 18 sub-areas. Then, in the test with the DNET trace, though locators in Random, PROPHET, and Cenwits have no or limited information about the mobility of the target nodes, they can meet the target nodes easily. DSearching has similar success rate in the tests with both traces because the locator always moves towards the most possible sub-area where the target node would be. Then, even when the number of sub-areas increases, it can stably find most target nodes. Such a result validates the effectiveness of the mobility information distribution in DSearching in networks with different sizes.

4.3.2 Average Delay

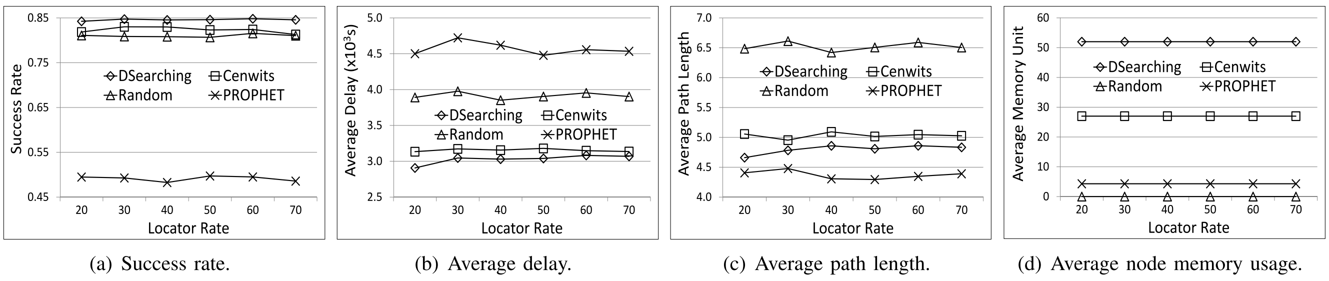
Figs. 7b and 8b present the average delays of the four methods in the tests with the DART trace and the DNET trace, respectively. We see that the average delays follow: DSearching < Cenwits < Random < PROPHET.

PROPHET has the highest average delay because the node followed by the locator may stay in certain sub-areas for a long time, resulting in an extremely long delay. Random also has a large average delay since nodes search randomly. Cenwits actively searches where the target node has shown recently. Since nodes usually would stay in a sub-area for a while, Cenwits has a small average delay. For DSearching, it further reduces the delay by utilizing both recent visiting records and long term MPT to guide node searching, which can help predict where the target node would be more accurately, leading to the least average delay.

4.3.3 Average Path Length

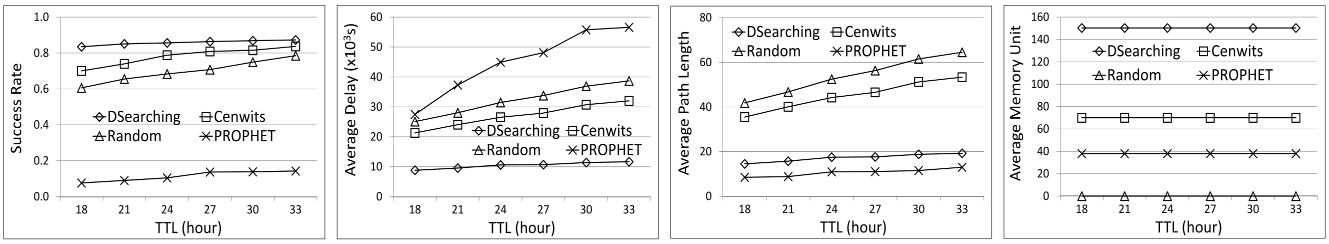
Figs. 7c and 8c show the average path lengths of the four methods in the tests with the DART trace and the DNET trace, respectively. We see that the average path lengths of the four methods follow: PROPHET < DSearching < Cenwits < Random.

PROPHET has the lowest average path length. This is because without direct information about the target's location, a locator can only follow the node that is more likely to meet the target node. Therefore, they often stay in a sub-area for a long time, rather than actively moving between sub-areas to search for the target node. Consequently, successful locators in PROPHET often only search a few sub-areas, leading to the shortest search path. DSearching has the second least average path length because the locator



(a) Success rate. (b) Average delay. (c) Average path length. (d) Average node memory usage.

Fig. 8. Performance with different locator rates using the DNET trace.



(a) Success rate. (b) Average delay. (c) Average path length. (d) Average node memory usage.

Fig. 9. Performance with different locator TTLs using the DART trace.

movement in it has the highest possibility of encountering the target node by utilizing both transient visiting records and long term mobility pattern of the target node. For Cenwits, it only utilizes the transient appearance records of the target node to guide the node searching, resulting in less efficient locator movement and longer average searching path length than DSearching. Random has the highest path lengths because the locator searches randomly in the network.

4.3.4 *Average Node Memory Usage*

Figs. 7d and 8d show the average node memory usages of the four methods in the tests with the DART trace and the DNET trace, respectively. We see that the average memory units of the four methods follow: Random $<$ PROPHET $<$ Cenwits $<$ DSearching.

Random has 0 average memory unit since nodes in it do not need to store any information for node searching. In PROPHET, each node needs to store its meeting probabilities with all other nodes, leading to a small amount of memory units. Cenwits has higher average memory units than PROPHET since it needs to store a large amount of node appearance records on each node. For DSearching, each node stores both visiting records and MPT of other nodes, resulting in the highest memory usage.

Though DSearching has the most memory usage among the four methods, the absolute amount of memory usage is acceptable. We find that the average memory unit on each node is about 150 and 50 in the tests with the two traces. Recall that each unit is about 50 bytes (i.e. one row of the MPT and four visiting records). This means that the average memory usage on each node is only about 7.5 and 2.5 KB in the two tests, which can easily be satisfied in modern devices. Therefore, we conclude that the designed mobility information distribution algorithm is memory efficient in DTNs.

We also see that the average memory units of the four methods are constant with different locator rates. This is because the memory usage on each node is only decided by node mobility and is irrelevant with the number of locators.

4.4 Experiments with Different Locator TTLs

We varied the TTL of each locator to see how different methods scale to locator TTL. Considering the DART trace is much longer than the DNET trace (119 days versus 20 days) and has relative slow node movement (student versus bus), we varied the TTL from 18 to 24 hours and from 2 to 7 hours for the DART trace and the DNET trace, respectively.

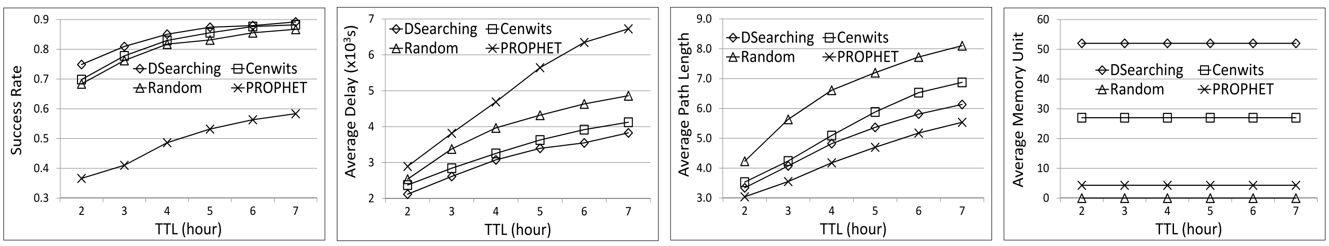
4.4.1 *Success Rate*

Figs. 9a and 10a present the success rates of the four methods in the tests with the DART trace and the DNET trace, respectively. We see that the success rates of the four methods follow: DSearching $>$ Cenwits $>$ Random $>$ PROPHET. This is the same as that in Figs. 7a and 8a for the same reasons.

We further find that when the TTL increases, Cenwits and PROPHET have closer and closer success rate with DSearching. This is because that the major difference between DSearching and the two methods is the node searching efficiency. When each locator is allowed to search longer, Cenwits and PROPHET can eventually find most target nodes, leading to a high success rate. However, this comes at the cost of increased average delay, as shown in the next section.

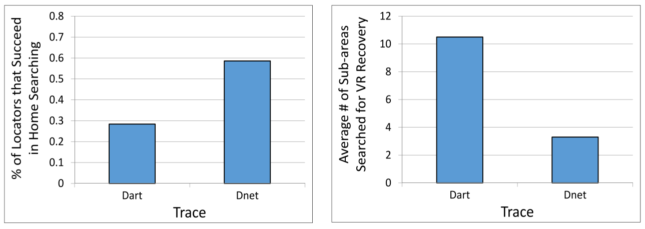
4.4.2 *Average Delay*

Figs. 9b and 10b present the average delays of the four methods in the tests with the DART trace and the DNET trace, respectively. We find that the average delays of the four methods follow the same relationship as in Figs. 7b and 8b: DSearching $<$ Cenwits $<$ Random $<$ PROPHET. The reasons are the same with those in the tests with different locator rates.



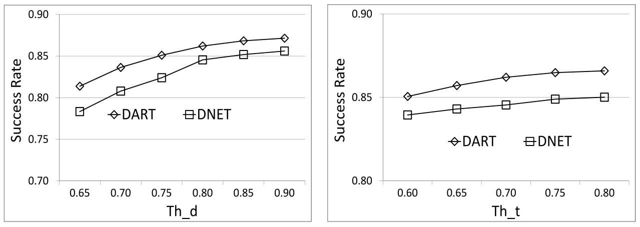
(a) Success rate. (b) Average delay. (c) Average path length. (d) Average node memory usage.

Fig. 10. Performance with different locator TTLs using the DNET trace.



(a) Efficiency of Home Searching. (b) Efficiency of VR Recovery.

Fig. 11. DSearching performance breakdown.



(a) Success rate with different Th_d. (b) Success rate with different Th_t.

Fig. 12. Influence of Th_d and Th_t.

We also see that when the TTL increases, the average delay increases. This is because when the TTL increases, locators that may fail to find their target nodes when the TTL is small can find their target nodes. Then, the average delay increases due to more successful locators with a large delay.

4.4.3 Average Path Length

Figs. 9c and 10c present the average path lengths of the four methods in the tests with the DART trace and the DNET trace, respectively. We find that the average path lengths follow: PROPHET < DSearching < Cenwits < Random, which is the same as in Figs. 7c and 8c. The reasons are also the same.

We also see that when TTL increases, the average path length increases. This is caused by the same reason as in Sections 4.4.1 and 4.4.2: when TTL increases, more locators can find their targets after searching many sub-areas, leading to increased average search path length.

4.4.4 Average Node Memory Usage

Figs. 9d and 10d present the average memory units a node uses in the tests with the DART trace and the DNET trace, respectively. We see that the results are the same with Figs. 7d and 8d. This is caused by the fact that the memory usage is independent with the locator TTL.

Combining all above results, we conclude that DSearching presents superior performance compared to other methods with different locator rates and TTLs. Such a result justifies our design goal: efficient node searching with acceptable cost.

4.5 Performance Breakdown

Recall that DSearching first searches the home sub-area of the target (Section 3.3.2) and then searches following the VR chain (Section 3.3.3). In the searching with VRs, when an expected VR cannot be found, the locator searches around to find a valid VR (Section 3.3.3). We further measure the effectiveness of each component to validate the design of DSearching. Specifically, we measured the percentage of successful locators that find their target nodes in home searching and in the searching with VRs, as well as the average number of sub-areas searched to recover a valid VR. In this test, we set the locator rate to 50 in both traces and TTL to 25 and 5 hours in the DART trace and the DNET trace, respectively. The test results on the two metrics are shown in Figs. 11a and 11b, respectively.

We see from Fig. 11a that the home searching step completes about 30 and 60 percent of successful locators in the tests with the DART trace and DNET trace, respectively. This shows that the design of home sub-area can effectively help find target nodes. However, it also shows that only searching the home sub-area is not enough. Other steps are needed for more efficient node searching. Such results validate the design of home searching and subsequent searching in DSearching.

We see from Fig. 11b that a locator needs to search about 10 and three sub-areas on average in order to recover a valid VR in the tests with the DART trace and DNET trace, respectively. The test with the DNET trace searches fewer sub-areas than that with the DART trace because there are much fewer sub-areas in the DNET trace. Such results demonstrate the efficiency of the proposed VR recovery method.

4.6 Influence of Th_d and Th_t

We further evaluated how two major parameters, i.e., Th_d and Th_t, affect the performance of DSearching. As introduced in Section 3.2, Th_d and Th_t determine how stably VRs and MPTs stay in their designated sub-areas to guide node searching, respectively. In this test, we varied Th_d from 0.65 to 0.9 and Th_t from 0.6 to 0.8, and other settings are the same as mentioned in Section 4.2. The test results are shown in Figs. 12a and 12b.

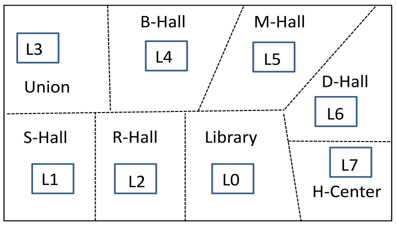
We see from Fig. 12a that when Th_d increases, the success rate of DSearching increases in the tests with both traces. This is because when Th_d increases, the locator can more easily find the expected VR when it arrives at a sub-area, thus can locate the target node more quickly. We also find that the success rate increases quickly when Th_d is low and becomes almost constant when Th_d is large (i.e., ≥ 0.8). This is because when Th_d is large, the locator can find the expected VR in most cases. Then, purely increasing Th_d cannot further increase the success rate significantly.

We see from Fig. 12b that the success rate of DSearching increases when Th_t increases. This is because when Th_t increases, the locator can more easily find the expected MPT information. We also find that the success rate only slightly increases when Th_t increases. This is because DSearching mainly relies on VR in node searching and only uses MPT when a valid VR cannot be found.

Combining above results, we conclude that the higher Th_d and Th_t are, the higher success rate DSearching can achieve. However, when they are larger than a threshold, only increasing the two thresholds cannot further enhance the searching efficiency. This validates our configuration of the two parameters in the paper.

4.7 Test with Real Environment Data

We further applied the DSearching to the mobility data of nine students on our campus. The nine students are from



(a) Maps for sub-area division.

Item	Value
# Sub-areas	8
# Nodes	9
# Transits	147
Duration	4 days

(b) Statistical data.

Fig. 13. Configuration in the real environment.

TABLE 7
Results with Real Environment Data

TTL (min)	20	30	40	50	60	70
Success Rate	0.62	0.75	0.83	0.88	0.89	0.93
Ave. Delay (min)	7.6	10.2	10.8	12.2	13.8	17.6
Ave. Path Length	1.4	1.9	2.0	2.3	2.6	3.1
Ave. Node Memory Usage	5	5	5	5	5	5

four departments in our university. We selected eight sub-areas, each of which is represented by a frequently visited building. Such mobility data is obtained based on the GPS records obtained by mobile phones, thus is more accurate than the AP association records in the two real traces. The test environment and the mobility information are summarized in Figs. 13a and 13b.

Since DSearching shows stable performance with different locator rates in previous tests, we only varied the locator TTL from 20 to 70 minutes and set the R_p to 40. Since the distance between the test buildings are not far, we assume that a locator averagely takes 5 minutes to move from one sub-area to a neighboring sub-area.

The test results are shown in Table 7. We find that when the locator TTL increases, success rate, average delay and average path length increase. This is the same as our observation in previous experiments with the two real traces. The reasons are the same that when the TTL increases, more locators can find the target nodes with a large delay and a long searching path.

We also see that when the TTL was set to 70 minutes, a successful locator takes only about 17 minutes (or three transits between sub-areas) to find the target node on average. Further, each node only needs 5 units of memory on average to support the node searching, which is very low and can easily be satisfied. In conclusion, DSearching is effective and efficient in searching mobile nodes in realistic DTNs.

4.8 Evaluation of the Advanced Extensions

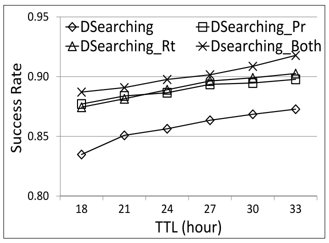
We further evaluate the effect of the extensions proposed in Section 3.5. Considering that DSearching presents stable performance in the tests with different locator rates, we only conducted tests with different TTLs. Specifically, as in Section 4.4, we again varied the TTL from 18 hours to 24 hours and from 2 to 7 hours for the DART trace and the DNET trace, respectively. Other settings are the same as mentioned in Section 4.2. In this test, we use DSearching to denote the original DSearching without any extensions, DSearching_Pr to denote DSearching with the improved MPT, DSearching_Rt to represent DSearching with the routine table, and DSearching_Both to represent DSearching with both the improved MPT and the routine table.

4.8.1 Success Rate

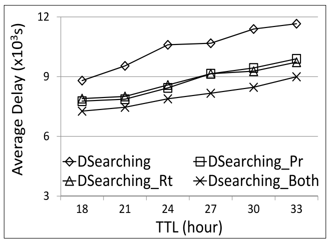
Figs. 14a and 15a present the success rates of DSearching and the three extensions in the tests with the DART trace and the DNET trace, respectively. We see from the two figures that in both tests, the success rates of the four methods follow: DSearching_Both $>$ DSearching_Rt $\approx$ DSearching_Pr $>$ DSearching. In DSearching_Pr, the advanced MPT can predict the next transit more accurately by considering the previous transit. In DSearching_Rt, the routine table can guide the locator to move to the places where the target visits regularly. Therefore, both DSearching_Pr and DSearching_Rt have higher success rate than the original DSearching. Since the two extensions improve the original DSearching from two different aspects, it is hard to distinguish which one is better. We can also see that they present similar success rate in our test. Furthermore, DSearching_-Both results in the highest success rate since it integrates both the improved MPT and the routine table.

4.8.2 Average Delay

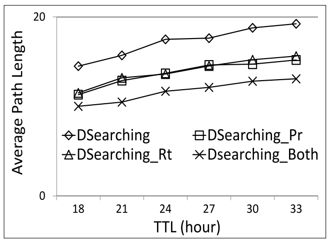
Figs. 14b and 15b show the average delays of DSearching and the three extensions in the tests with the DART trace and the DNET trace, respectively. We find from the two figures that the average delays of the four methods follow DSearching_Both $<$ DSearching_Rt $\approx$ DSearching_Pr $<$ DSearching. Such a result matches with the relationship on success rate of each method, as shown in the previous section. This is because by utilizing the improved MPT and the routine table, DSearching_Rt and DSearching_Pr can enable locaters to find the target more quickly, thereby reducing the average search delay. Similarly, DSearching_Both has the lowest average delay because it enables both the



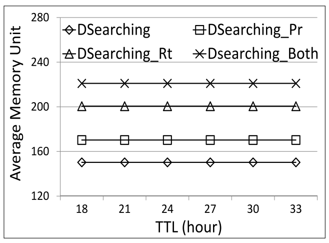
(a) Success rate.



(b) Average delay.

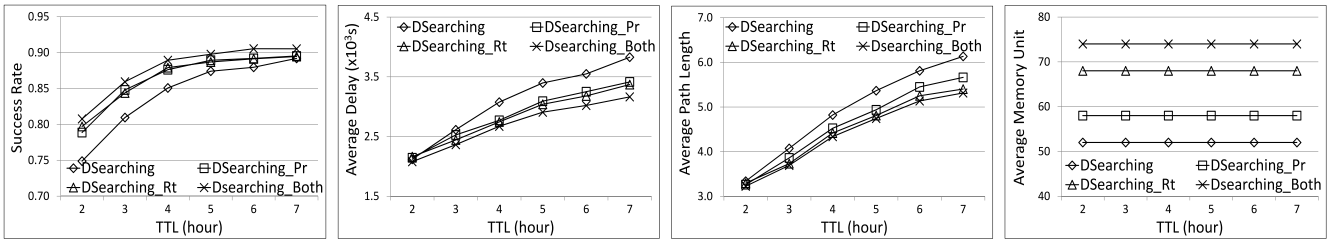


(c) Average path length.



(d) Average node memory usage.

Fig. 14. Performance of each extension with different locator TTLs using the DART trace.



(a) Success rate. (b) Average delay. (c) Average path length. (d) Average node memory usage.

Fig. 15. Performance of each extension with different locator TTLs using the DNET trace.

improved MPT and the routine table, both of which can improve node searching efficiency.

4.8.3 Average Path Length

Figs. 14c and 15c show the average path lengths of DSearching and the three extensions in the tests with the DART trace and the DNET trace, respectively. We observe from the two figures that the average path lengths of the four methods follow DSearching_Both $<$ DSearching_Rt $\approx$ DSearching_Pr $<$ DSearching. In DSearching_Pr, the improved MPT can guide the locator to search along the path of the target more accurately. In DSearching_Rt, the routine table can guide the locator to move to the regularly visited places of the target node. Both extensions effectively avoid searching unnecessary sub-areas. Thereby, they lead to shorter average path length than DSearching. Again, DSearching_Both leads to the shortest average search path because both extensions are included.

Combining above results, we conclude that the two extensions can improve the node searching efficiency.

4.8.4 Average Node Memory Usage

Figs. 14d and 15d illustrate the average node memory usages of DSearching and other three extensions in the tests with the DART trace and the DNET trace, respectively. We see from the two figures that the average node memory usage of the four methods follow DSearching_Both $>$ DSearching_Rt $>$ DSearching_Pr $>$ DSearching. In DSearching_Pr, the MPT table is extended to include more fine-grained transit probabilities, resulting in more memory usage than DSearching. For DSearching_Rt, the routine table is created and distributed in the same way as the MPT table. Therefore, it generates even more memory usage than DSearching_Pr. Then, DSearching_Both has the highest node memory usage by including both extensions.

Though these extensions lead to more memory usage, they bring about improvement on the node searching efficiency. On the other hand, the increased memory usage is less than 50 percent of the original memory usage. This means that the two extensions do not lead to a significant burden on memory usage. The system designer can decide which extensions can be enabled based on the actual system need to achieve a balance on efficiency and memory usage.

5 Conclusion

In this paper, we propose DSearching, a distributed mobile node searching scheme in DTNs where nodes present certain mobility patterns. In DSearching, the whole network is split into sub-areas. A node's mobility information includes both transient sub-area visiting records and long-term mobility pattern between sub-areas. Each node distributes its visiting record for a sub-area to nodes that are likely to stay in the previous sub-area. The long term mobility information of a node in a subarea is distributed to a limited number of long-staying nodes in the sub-area. Such information enables the locator to search along the path the target node traverses, resulting in efficient node searching. Advanced extensions that utilize sub-area visiting correlation and routine information are also proposed in the paper to further enhance the node searching efficiency. Extensive experiments with both real traces and on-campus DTN trace validate the high effectiveness and high efficiency of DSearching. In the future, we plan to investigate how to fully utilize searching agents to further improve node searching efficiency.

Acknowledgments

This research was supported in part by the US National Science Foundation (NSF) grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, Microsoft Research Faculty Fellowship 8300751. An early version of this work was presented in the Proceedings of INFOCOM 2014 [16]. Haiying Shen is the corresponding author.

References

[1] S. Jain, K. R. Fall, and R. K. Patra, "Routing in a delay tolerant network," in *Proc. Conf. Appl., Technol., Archit. Protocols Comput. Commun.*, 2004, pp. 145–158.

[2] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet," in *Proc. 10th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2002, pp. 96–107.

[3] J. Zhao, Y. Zhu, and L. M. Ni, "Correlating mobility with social encounters: Distributed localization in sparse mobile networks," in *Proc. IEEE 9th Int. Conf. Mobile Adhoc Sensor Syst.*, 2012, pp. 10–18.

[4] S. Lu, Y. Liu, Y. Liu, and M. Kumar, "Loop: A location based routing scheme for opportunistic networks," in *Proc. IEEE 9th Int. Conf. Mobile Adhoc Sensor Syst.*, 2012, pp. 118–126.

[5] K. Chen and H. Shen, "Leveraging social networks for P2P content-based file sharing in disconnected Manets," *IEEE Trans. Mobile Comput.*, vol. 13, no. 2, pp. 235–249, Feb. 2014.

[6] B. Thorstensen, T. Syversen, T. Walseth, and T.-A. Bjørnvold, "Electronic shepherd-a low-cost, low-bandwidth, wireless network system," in *Proc. 2nd Int. Conf. Mobile Syst., Appl. Services*, 2004, pp. 245–255.

[7] J. Huang, S. Amjad, and S. Mishra, "CenWits: A sensor-based loosely coupled search and rescue system using witnesses," in *Proc. 3rd Int. Conf. Embedded Netw. Sensor Syst.*, 2005, pp. 180–191.

[8] J.-H. Huang, L. Jiang, A. Kamthe, J. Ledbetter, S. Mishra, A. Cerpa, and R. Han, "SenSearch: GPS and witness assisted tracking for delay tolerant sensor networks," in *Proc. 8th Int. Conf. Ad-Hoc, Mobile Wireless Netw.*, 2009, pp. 255–269.

[9] A. Balasubramanian, B. N. Levine, and A. Venkataramani, "DTN routing as a resource allocation problem," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2007, pp. 373–384.

[10] 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.

[11] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: social-based forwarding in delay tolerant networks," in *Proc. 9th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2008, pp. 241–250.

[12] E. M. Daly and M. Haahr, "Social network analysis for routing in disconnected delay-tolerant Manets," in *Proc. 8th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2007, pp. 32–40.

[13] J. Link, D. Schmitz, and K. Wehrle, "GeoDTN: Geographic routing in disruption tolerant networks," in *Proc. IEEE Global Telecommn. Conf.*, 2011, pp. 1–5.

[14] I. Leontiadis and C. Mascolo, "GeOpps: Geographical opportunistic routing for vehicular networks," in *Proc. IEEE Int. Symp. World Wireless Mobile Multimedia Netw.*, 2007, pp. 1–6.

[15] L. Song, U. Deshpande, U. C. Kozat, D. Kotz, and R. Jain, "Predictability of WLAN mobility and its effects on bandwidth provisioning," in *Proc. IEEE INFOCOM*, 2006, pp. 1–13.

[16] K. Chen, H. Shen, and L. Yan, "DSearching: Distributed searching of mobile nodes in DTNs with floating mobility information," in *Proc. INFOCOM*, 2014, pp. 2283–2291.

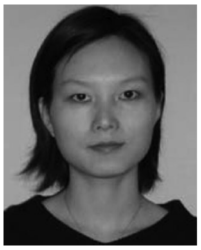
[17] T. Henderson, D. Kotz, and I. Abyzov, "The changing usage of a mature campus-wide wireless network," in *Proc. 10th Annu. Int. Conf. Mobile Comput. Netw.*, 2004, pp. 187–201.

[18] K. Chen and H. Shen, "DTN-FLOW: Inter-landmark data flow for high-throughput routing in DTNs," in *Proc. IEEE 27th Int. Symp. Parallel Distrb. Symp.*, 2013, pp. 726–737.

[19] A. Balasubramanian, B. N. Levine, and A. Venkataramani, "Enhancing interactive web applications in hybrid networks," in *Proc. 14th ACM Int. Conf. Mobile Comput. Netw.*, 2008, pp. 70–80.

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

Kang Chen received the BS degree in electronics and information engineering from Huazhong University of Science and Technology, China, in 2005, the MS degree in communication and information systems from the Graduate University of Chinese Academy of Sciences, China in 2008, and the PhD degree in computer engineering from Clemson University. He is currently a postdoctoral research fellow in the Department of Electrical and Computer Engineering, Clemson University. His research interests include delay tolerant networks, vehicular 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 PhD degrees in computer engineering from Wayne State University in 2004 and 2006, respectively. She is currently an associate professor in the Department of Electrical and Computer Engineering, Clemson University. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing. She is a Microsoft faculty Fellow of 2010, a senior member of the IEEE, and a member of the ACM.

Li Yan received the BS degree in information engineering from Xi'an Jiaotong University, China, in 2010, and the MS degree in electrical engineering from University of Florida in 2013. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering, Clemson University, South Carolina, United States. His research interests include wireless networks, with an emphasis on delay tolerant networks and sensor networks.