ISSN:2348-2079

Volume-5 Issue-2

International Journal of Intellectual Advancements and Research in Engineering Computations

Leveraging Social Networks for P2P Content-Based File Sharing in Disconnected MANETs

C.MANI M.C.A., M.Phil.,ME Associate Professor Department Of MCA

V.Kunalakshmi Final M.C.A

Nandha Engineering College Erode

Mail Id- kuna.sv165@Gmail.Com

1 INTRODUCTION

In the past few years, personal mobile devices such as laptops, PDAs, and smartphones have been more and more popular. Indeed, the number of smartphone users increased by 118 million across the world in 2007 [1], and is expected to reach around 300 million by 2013 [2]. The incredibly rapid growth of mobile users is leading to a promising future, in which they can freely share files between each other whenever and wherever. The number of mobile searching users (through smartphones, feature phones, tablets, etc.) is estimated to reach 901.1 million in 2013 [3]. Currently, mobile users interact with each other and share files via an infrastructure formed by geographi-cally distributed base stations. However, users may find themselves in an area without wireless service (e.g., mountain areas and rural areas). Moreover, users may hope to reduce the cost on the expensive infrastructure network data.

The P2P file sharing model makes large-scale networks a blessing instead of a curse, in which nodes share files directly with each other without a centralized server. Wired

P2P file sharing systems (e.g., BitTorrent [4] and Kazaa [5]) have already become a popular and successful paradigm for file sharing among millions of users. The successful deployment of P2P file sharing systems and the aforemen-tioned impediments to file sharing in MANETs make the P2P file sharing over MANETs (P2P MANETs in short) a promising complement to current infrastructure model to realize pervasive file sharing for mobile users. As the mobile digital devices are carried by people that usually belong to certain social relationships, in this paper, we focus on the P2P file sharing in a disconnected MANET community consisting of mobile users with social network properties. In such a file sharing system, nodes meet and exchange requests and files in the format of text, short videos, and voice clips in different interest categories. A typical scenario is a course material (e.g., course slides, review sheets, assignments) sharing system in a school campus. Such a scenario ensures for the most that nodes sharing the same interests (i.e., math), carry corresponding files (i.e., math files), and meet regularly (i.e., attending math classes).

In MANETs consisting of digital devices, nodes are constantly moving, forming disconnected MANETs with opportunistic node encountering. Such transient network connections have posed a challenge for the development of P2P MANETs. Traditional methods supporting P2P MANETs are either flooding-based [6], [7], [8], [9] or advertisement-based [10], [11], [12]. The former methods rely on flooding for file searching. However, they lead to

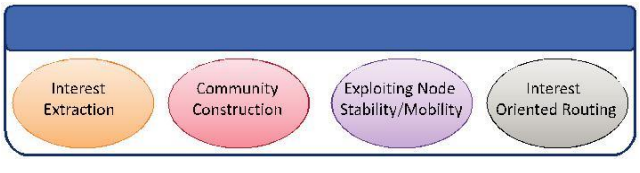


Fig. 1. Components of SPOON.

high overhead in broadcast. In the latter methods, nodes advertise their available files, build content tables, and forward files according to these tables. But they have low search efficiency because of expired routes in the content tables caused by transient network connections. Also, advertising can lead to high overhead.

Some researchers [13], [14], [15], [16], [17] further proposed to utilize cache/replication to enhance data dissemination/access efficiency in disconnected MANETs. However, nodes in these methods passively wait for contents that they are interested in rather than actively search files, which may lead to a high search delay.

Recently, social networks are exploited to facilitate content dissemination/publishing in disconnected MANETs [18], [19], [20], [21]. These methods exploit below property to improve the efficiency of message forwarding:

. (P1) nodes (i.e., people) usually exhibit certain movement patterns (e.g., local gathering, diverse centralities, and skewed visiting preferences).

However, these methods are only for the dissemination of information to subscribers. They are not specifically designed for file searching. Also, they fail to take into account other properties of social networks revealed by recent studies to facilitate content sharing:

. (P2) Users usually have a few file interests that they visit frequently [22] and a user's file visit pattern follows a power-law distribution [23].
. (P3) Users with common interests tend to meet with each other more often than with others [24].

By leveraging these properties of social networks, we propose social network-based P2P cOntent-based file sharing in disconnected mObile ad hoc Networks (SPOON) with four components as shown in Fig. 1:

1. Based on P2, we propose an interest extraction algorithm to derive a node's interests from its files. The interest facilitates queries in content-based file sharing and other components of SPOON.
2. We refer to a collective of nodes that share common interests and meet frequently as a community. According to P3, a node has high probability to find interested files in its community. If this fails, based on P1, the node can rely on nodes that frequently travel to other communities for file searching. Thus, we propose the community construction algorithm to build communities to enable efficient file retrieval.
3. According to P1, we propose a node role assignment algorithm that takes advantage of node mobility for efficient file searching. The algorithm designates a stable node that has the tightest connections with others in its community as the community coordinator to guide intracommunity searching. For each known foreign community, a node that frequently travels to it is designated as the commu-nity ambassador for intercommunity searching.
4. We propose an interest-oriented file searching and retrieval scheme that utilizes an interest-oriented routing algorithm (IRA) and above three components. Based on P3, IRA selects forwarding node by considering the probability of meeting interest keywords rather than nodes. The file searching scheme has two phases: Intra- and intercommunity searching. In the former, a node first queries nearby nodes, then relies on coordinator to search the entire home commu-nity. If it fails, the intercommunity searching uses an ambassador to send the query to a matched foreign community. A discovered file is sent back through the search path or the IRA if the path breaks.

SPOON is novel in that it leverages social network properties of both node interest and movement pattern. First, it classifies common-interest and frequently encoun-tered nodes into social communities. Second, it considers the frequency at which a node meets different interests rather than different nodes in file searching. Third, it chooses stable nodes in a community as coordinators and highly mobile nodes that travel frequently to foreign communities as ambassadors. Such a structure ensures that a query can be forwarded to the community of the queried file quickly. SPOON also incorporates additional strategies for file prefetching, querying-completion and loop-preven-tion, and node churn consideration to further enhance file searching efficiency.

The rest of the paper is arranged as follows: Section 2 provides an overview of related works. Section 3 presents the design of the components of SPOON. In Section 4, the performance of SPOON is evaluated in comparison with other systems. The last section presents concluding remarks and future work.

2 Related Work

2.1 P2P File Sharing in MANETs

We first introduce the P2P file sharing algorithms designed in MANETs.

2.1.1 Flooding-Based Methods

In flooding-based methods, 7DS [6] is one of the first approaches to port P2P technology to mobile environ-ments. It exploits the mobility of nodes within a geo-graphic area to disseminate web content among neighbors. Passive distributed indexing (PDI) [8] is a general-purpose distributed file searching algorithm. It uses local broad-casting for content searching and sets up content indexes on nodes along the reply path to guide subsequent searching. Klemm et al. [7] proposed a special-purpose on-demand file searching and transferring algorithm based on an application layer overlay network. The algorithm transparently aggregates query results from other peers to eliminate redundant routing paths. Hayes [9] extended the Gnutella system to mobile environments and proposed the use of a set of keywords to represent user interests.

However, these flooding-based methods produce high overhead due to broadcasting.

2.1.2 Advertisement-Based Methods

Tchakarov and Vaidya [10] proposed GCLP for efficient content discovery in location-aware ad hoc networks. It disseminates contents and requests in crossed directions to ensure their encountering. P2PSI [11] combines both adver-tisement (push) and discovery (pull) processes. It adopts the idea of swarm intelligence by regarding shared files as food sources and routing tables as pheromone. Each file holder regularly broadcasts an advertisement message to inform surrounding nodes about its files. The discovery process locates the desired file and also leaves pheromone to help subsequent search requests. Repantis and Kalogeraki [12] proposed a file sharing mechanism in which nodes use the Bloom filter to build content synopses of their data and adaptively disseminate them to other nodes to guide queries. Though the advertisement-based methods reduce the over-head of flooding-based methods, they still generate high overhead for advertising and cannot guarantee the success of file searching due to node mobility.

2.2 P2P File Sharing in Disconnected MANETs

The disconnected MANETs are featured by sparse node density and intermittent node connection, which makes previously introduced methods infeasible in such networks. We then further introduce two categories of P2P file sharing methods for disconnected MANETs.

2.2.1 Cache/Replication-Based Methods

Huang et al. [13] proposed a method that considers multiple factors (e.g., node mobility, file popularity, and file server topology) in creating file replicas in file servers to realize optimal file availability in content distribution community. Gao et al. [14] proposed cooperative caching in disruption tolerant networks. It replicas each file to network central locations, which are frequently visited by nodes in the system, to ensure efficient data access. QCR [15] uses file caching to realize effective multimedia content dissemina-tion in opportunistic networks. In addition to node mobility and file popularity, it also considers the impatience of users when creating replicas

2.2.2 Social Network-Based Methods

Recently, social networks have been utilized in content publishing/dissemination algorithms [18], [19], [20], [21] in opportunistic networks. MOPS [18] provides content-based sub/pub service by utilizing the long-term neighboring relationship between nodes. It groups nodes with frequent contacts and selects nodes that connect different groups as brokers, which are responsible for intercommunity communication. Then, contents and subscriptions are relayed through brokers to reach different communities. MOPS only considers node mobility, while SPOON is more advanta-geous by considering both node interest and mobility as described previously. Moreover, unlike MOPS that only depends on the meeting of brokers for intercommunity search, SPOON enhances the efficiency of intercommunity search by 1) assigning one ambassador for each known foreign community, which helps to forward a query directly to the destination community, and 2) utilizing stable nodes (coordinator) to receive messages from ambassadors.

3 The Design of SPOON

In this section, we first present trace data analysis to verify the social network properties in a real MANET. A P2P MANET file sharing system usually consists of 1) a method to represent contents, 2) a node management structure, and 3) a file searching method based on steps 1 and 2. Accordingly, SPOON has three main components: 1) interest extraction, 2) structure construction including community structure and node role assignment, and 3) interest-oriented file searching and retrieval based on components 1 and 2. We then present each component of SPOON.

3.1 Trace Data Analysis

To validate the correlation between node interests and their contact frequencies, we analyzed the trace from the Haggle project [26], which contains the encountering records among 98 mobile devices carried by scholars attending the Infocom'06 conference. Some participants completed questionnaires, indicating the conference tracks that they are interested in.

We use T_t to denote the time length of the trace, and define the total meeting time of two nodes as the sum of the time length of each encountering. By regarding a community as a group of nodes in which each node has total meeting time larger than T_t=4 with at least half of all nodes in the community, we detected eight communities from the trace. We then calculated each node's average number of shared interested tracks with other members in

TABLE 1
Average Number of Shared Interested Tracks

Community C_i	Ave. # of shared interests with nodes in C_i	Ave. # of shared interests with nodes not in C_i
1	1.50	0.99
2	0.83	0.69
3	1.17	0.79
4	1	0.39
5	1.93	0.94
6	0.33	0.21
7	1.1	0.71
8	1	0.33

TABLE 2
Notations in Interest Extraction

Notation	Meaning
f_i and G_u	the i-th file and u-th interest group in a node
w_{it_k} and $\bar{w}_{ut_k}$	the weight of keyword t_k in f_i and in G_u
f_{ui}	the i-th file in G_u
v_i	the file vector of f_i
$\bar{v}_u$	the group vector of G_u
$\bar{v}_N$	the node vector of node N

its own community C_i ($0 < i < 8$), and with nodes in all other communities, respectively. Finally, the average values of all nodes in each community are calculated and shown in Table 1.

From the table, we see that for each community, nodes have higher average number of shared interested tracks with same community nodes than with nodes from other communities. Note that we used a relatively loose com-munity creation requirement that each node only needs to have a high contact frequency with half of nodes in a community. With a stricter requirement and a more sophisticated clustering method, nodes in the same com-munity would share more interested tracks. Above traces verify the previously observed social properties and support the basis for SPOON that nodes with common interests tend to meet frequently.

3.2 Interest Extraction

Without loss of generality, we assume that node contents can be classified to different interest categories. It was found that users usually have a few file categories that they query for files frequently in a file sharing system. Specifically, for the majority of users, 80 percent of their shared files fall into only 20 percent of total file categories [22]. Like other file sharing systems [27], [28], we consider that a node's stored files can reflect its file interests. Thus, SPOON derives the interests of a node from its files. Table 2 lists the notations used in this section.

To derive its interests, a node infers keywords from each of its files using the document clustering technique [29]. Specifically, a node derives a file vector for each of its files from its metadata. For file f_i, we denote its file vector by $v_i = (t_1, w_{it_1}; t_2, w_{it_2}; t_3, w_{it_3}; \ldots; t_m, w_{it_m})$, in which t_k and w_{ik} ($1 \le k \le m$) denote a keyword and its weight that represents the importance of the keyword in describing the file. We adopt the method in the text retrieval literature [30] to calculate the weight of a keyword, say t_k, in a file, say f_i, with below formula:

$$w_{itk} = 1 + \log(n_{tk}), \quad (1)$$

where n_{tk} refers to the number of occurrences of keyword t_k. Suppose there are m keywords in the file, we further normalize the weights by

$$w_{itk} = w_{itk} \Big/ \sqrt{\sum_{q=1}^{m} w_{itq}}. \quad (2)$$

Then, to calculate the similarity of two file vectors, say $v_1 = (t_1, w_{1t_1}; t_2, w_{1t_2}; t_4, w_{1t_4})$ and $v_2 = (t_1, w_{2t_1}; t_3, w_{2t_3}; t_5, w_{2t_5})$, we first generate their common vector, which consists of their common keywords and corresponding weights in their own vectors. For example, the common vector of v_1 and v_2 is (t_1, w_{1t_1}) from v_1 and (t_1, w_{2t_1}) from v_2. We then use the following formula to calculate the similarity between v_1 and v_2:

$$sim(v_1, v_2) = \frac{\sum_{k=1}^{m'} w_{1k} \cdot w_{2k}}{\sqrt{\sum_{k=1}^{m'} w_{1k}^2} \cdot \sqrt{\sum_{k=1}^{m'} w_{2k}^2}}, \quad (3)$$

where m' is the total number of common keywords and w_{1k} and w_{2k} represent the weights of the kth common keyword of the two vectors, respectively.

After retrieving the file vector of each of its files, a node classifies its files to derive its interest groups. It creates a file similarity matrix $A = sim(v_r, v_s)$ ($1 \le r, s \le m$), where m is the number of files the node has. Since the similarities among files are known, we use the AGNES method [31] to cluster the files into interest groups in a hierarchical manner. Each file forms an individual group initially. Then, two most similar file groups are merged in each step. This process repeats until the similarity between any two groups is below a threshold. The similarity between two groups is calculated based on their interest vectors introduced below. Consequently, a file is classified to only one interest group and there is no overlap among groups.

Each group has a number of files. Suppose there are g files in interest group G_u, denoted by $(f_{u1}, f_{u2}, \ldots, f_{ug})$. The average weight of a keyword, say t_k, in the group is calculated by $w_{utk} = \sum_{i=1}^{g} w_{tk}^{u_i} / g$, where $w_{tk}^{u_i}$ denotes the weight of t_k in f_{ui}. We also predefine a threshold for the average weight, denoted by T_w. We form an interest vector with keywords having weights larger than T_w and use it to represent interest group G_u:

$$v_u = (t_1, w_{ut_1}; t_2, w_{ut_2}; t_3, w_{ut_3}; \ldots; t_{n'}, w_{tun'}). \quad (4)$$

where n' is the total number of keywords in v_u. Thus, each node has a number of interest vectors to represent its interests. The weight of G_u, denoted $W(G_u)$, is the portion of the group's files in all files of the node. We then generate a node vector (v_N) to describe a node's interests. The keywords of v_N is the keyword union of all interest group vectors, and the weight of each keyword is the sum of its weights in different interest groups it belongs to normalized by the weights of these groups.

3.3 Community Construction

Social network theory reveals that people with the same interest tend to meet frequently [24]. By exploiting this property, SPOON classifies nodes with common interests and frequent contacts into a community to facilitate

interest-based file searching, as introduced latter in Section 3.5. Nodes with multiple interests belong to multiple communities. The community construction can easily be conducted in a centralized manner by collecting node interests and contact frequencies from all nodes to a central node. However, considering that the proposed system is for distributed disconnected MANETs, in which timely information collection and distribution is nontrivial, we further propose a decentralized method to ensure the adaptivity of SPOON in real environment.

When two nodes, say N_1 and N_2, meet, they consider two cases for community creation: 1) they do not belong to any communities, and 2) at least one of them is already a member of a community. In the first case, they calculate the similarity between each pair of their interest vectors using

(3). A pair of interest groups, say G_i and G_j with interest vectors v_i and v_j, is called matched interest group when W ðG$_i$ÞW ðG$_j$Þsimðv$_i$; v$_j$Þ T_G, where T_G is a predefined threshold. The purpose of taking into account the weight of each interest group is to eliminate the noise of interest groups with a small number of files and achieve better interest clustering. If N_1 and N_2 have at least one pair of matched interest group, and their contact frequency, F ðN$_1$; N$_2$Þ, is higher than the top h_1 percent highest encountering frequencies in either node, the two nodes form a new community. The keywords in their matched interest groups and corresponding weights constitute the community vector (v_C) of the community.

In the second case, suppose N_2 is already a member of community C, N_1 calculates WðG$_i$ Þsimðv$_i$; v_C Þ for each of its interest groups, say G_i, to decide whether it should join in community C. If the similarity value for one interest group is larger than T_G, and N_1's contact frequency with community C is higher than the top h_2 percent of N_1's contact frequencies with all nodes it has met, N_1 is granted the membership to community C. The contact frequency with community C refers to the accumulated contact frequency with nodes in C. It is updated upon each encountering with a node in C. This means that N_i contacts members in community C frequently enough to guarantee connections. N_1 then copies the community vector and other community information from N_2. Also, when a node meets the community coordinator, it reports its files to the coordinator to update its file index and community vector. The coordinator then forwards the updated community information to community members when meets them.

With above community construction method, nodes with common interests and frequent contacts gradually form a community. However, nodes that appear later have more stringent community acceptance requirement. Its contact frequency to the community needs to be higher than that of more nodes, and its interest vector is compared with a longer community vector. Also, nodes in a group admit new members distributively. As a result, nodes in a group may not have very similar interests or high contact frequencies. We propose two solutions to alleviate this problem. First, we set an initial period for newly joined nodes in which they accumulate contact frequencies with others. Then, when a node starts to join in communities, its meeting frequencies with others are relatively stable, which provides more accurate measurement for determining the communities to join in. Second, we use group member pruning. Existing community members can have a second round voting to confirm the eligibility of new community members. Specifically, if N_2 in community C finds a node, say N_1, satisfies the requirements of C, it awards N_1 a potential membership for C. Then, other community members in C further checks N_1's eligibility to join in C. That is, every time when N_1 meets an existing member of C, say N_3, N_3 checks whether they have at least one pair of matched interest group and whether their contact frequency is higher than the top h_1 percent of N_1's highest contact frequencies. If yes, N_3 approves the membership of N_1. When an existing community member of C notices that N_1 receives the grants from half of the community members, it grants N_1 the group membership.

Another issue is that node contact frequencies and interests may change over time. Since the community construction algorithm is continuous running, a node can detect that it fails to satisfy the requirement of current community. It then withdraws from the current commu-nity, notifies connected nodes in it, and searches for a new community to join in.

The values of the thresholds used in the community construction process (T_G, h_1, h_2) are determined by many factors such as number of nodes, number of interests, and types of applications. Generally, T_G decides the interest tightness among nodes in each community. A larger T_G leads to higher similarity between interests of nodes in one community, but also generates more communities. Therefore, T_G should be configured based on application scenario. If the application has clear file categories (i.e., course file sharing), we can set a large T_G to gather files in the same category. Otherwise, a medium T_G should be set to balance the interest closeness and the number of commu-nities. h_1 and h_2 determine the tightness of a community. The smaller h_1 and h_2 are, the tighter the community is. Therefore, we set them to 30 by default in experiment to ensure frequent contact among community members. These values were set based on empirical experiences. We leave further investigation on appropriate values as future work.

3.4 Node Role Assignment

A previous study has shown that in a social network consisting of mobile users, only a part of nodes have high degrees [20]. We can often find an important or popular person who coordinates members in a community in our daily life. For example, the college dean coordinates different departments in the college, and the department head connects to faculty members in the department. Thus, we take advantage of different types of node mobility for file sharing.

We define community coordinator and ambassador nodes in the view of a social network. A community coordinator is an important and popular node in the community. It keeps indexes of all files in its community. Each community has one ambassador for each known foreign community, which serves as the bridge to the community. The coordinator in a community maintains the v_C of foreign communities and corresponding ambassadors to map queries to ambassadors for intercommunity

searching. The number of ambassadors and coordinators can be adjusted based on the network size and workload to avoid overloading these nodes. Since ambassadors and coordinators take more responsibility, we can also adopt role rotation and extra incentives for fairness consideration. Due to page limit, we leave this as our future work.

3.4.1 Community Coordinator Node Selection

We define a stable node that has tight contact frequency with other community members as the community coordi-nator. In network analysis, centrality is often used to determine the relative importance of a vertex within the network. We then adopt the improved degree centrality [32], which assigns weight to each link based on the contact frequency, for coordinator selection because it reflects the tightness of a node with other community members. In the initial phase of coordinator discovery, each node, say node N_i, in a community collects contact information from its neighbors in the same community and then calculates its degree centrality by

$$D_{p_i} = \sum_{j=1, j\neq i}^{N} w_{ij}, \quad (5)$$

where w_{ij} is the link weight between N_i and N_j and N is the number of neighbors in the same community. To reflect the property that the coordinator has the most connections with all community members, w_{ij} equals 1 if the contact frequency between N_i and N_j is larger than a threshold and 0 otherwise. Though such a method cannot ensure its connection to every community member, it ensures that the coordinator has the tightest overall connection to all community members.

Each node periodically checks its degree centrality and broadcasts such information to all community members. If a node receives no larger centrality score than its own centrality for three consecutive periods, it claims itself as the potential coordinator. The potential coordinator would confirm its status as the coordinator when meets the previous one. If it is confirmed, it then requests the community information from the old coordinator. Also, when the new coordinator meets community members, they exchange information for group vector update and ambassador selection, as well as request routing.

3.4.2 Community Ambassador Node Selection

An ambassador is used to bridge the coordinator in its home community and a foreign community. We use the product of a node's contact frequency with its coordinator and that with the foreign community for ambassador selection. Each node i calculates its utility value for foreign community k by

$$U_{ik} = F(N_i, C_k) \cdot F(N_i, N_c), \quad (6)$$

where C_k represents foreign community k, N_c is the coordinator in its home community, and $F(\cdot)$ denotes the meeting frequency. Each node reports its utility values for foreign communities it has met to the coordinator in its home community. Then, the community coordinator chooses one ambassador for each known foreign community. Also, the node that has the highest overall contact frequency with all foreign communities is selected as the default ambassa-dor. In case that a request fails to find a matched ambassador, the default ambassador can carry the request and seek for potential forwarders in foreign communities. If an ambassador loses the connection with the coordinator for a certain period of time, a new ambassador that satisfies above requirements is selected. This arrangement facilitates interest-oriented file searching by enabling a coordinator to send file requests to matched foreign communities quickly.

In above design, ambassadors are the key to connect different communities efficiently. Coordinators achieve balance between the centralized and distributed searching by checking whether a community can satisfy a query quickly, which is important in disconnected MANETs. Also, though broadcast is used in coordinator selection, the cost is limited because 1) it is only among community members, and 2) we can set a long interbroadcast period because nodes usually have stable degree centrality. To select ambassadors, each node just reports its utility values to the coordinator, which can be piggybacked on the beacon messages. Therefore, this step does not incur significant extra costs.

3.5 Interest-Oriented File Searching and Retrieval

In social networks, people usually have a few file interests [22] and their file visit pattern generally follows a certain distribution [23]. Also, people with the same interest tend to contact each other frequently [24]. Thus, interests can be a good guidance for file searching. Considering the relation among node movement pattern [33], individuals' common interests, and their contact frequencies, we can route file requests to file holders based on nodes' frequencies of meeting different interests.

Then, the interest-oriented file searching scheme has two steps: intracommunity and intercommunity searching. A node first searches files in its home community. If the coordinator finds that the home community cannot satisfy a request, it launches the intercommunity searching and forwards the request to an ambassador that will travel to the foreign community that matches the request's interest. A request is deleted when its time-to-live (TTL) expires. During the search, a node sends a message to another node using the interest-oriented routing algorithm, in which a message is always forwarded to the node that is likely to hold or to meet the queried keywords. The retrieved file is routed along the search path or through IRA if the route expires.

3.5.1 Interest-Oriented Routing Algorithm

In SPOON, every node maintains a history vector that records its frequency of encountering interest keywords. The history vector is in the form of $v_H = (t_0, w_{h0}, t_1, w_{h1}, t_2, w_{h2}, \ldots, t_n, w_{hn})$, where w_{hi} is the aggregated times of encountering keyword t_i. w_{hi} decays periodically as time passes by $w_{hi} = w_{hi}(\delta < 1)$. When two nodes meet, they exchange their node vectors and update history vectors. The history vector is used to evaluate the probability of meeting the queried content.

The destination of a request is represented by a vector $v_{dest} = (t_0, w_0, t_1, w_1, t_2, w_2, \ldots, t_n, w_n)$. In IRA, a node uses

the fitness score F to evaluate its neighbors' probabilities to be or to meet the file holder. The fitness F of neighbor i is measured by $F = sim(v_{dest}, \vec{v}_i) + sim(v_{dest}, v_{Hi})$, where $\vec{v}_i$ and v_{Hi} are the node vector and history vector of node i, respectively. The factor of $sim(v_{dest}, \vec{v}_i)$ aims to find the node sharing the most similar interests with the destination, and the factor of $sim(v_{dest}, v_{Hi})$ aims to find a node that is very likely to meet the destination in its movement. is used to control the weight of these two factors. In IRA, when a node receives a message, if its neighbor with the highest F has higher F than itself, it forwards the message to the neighbor. This process repeats until the message arrives at the destination. Coordinators do not use IRA but send messages to its community members when meeting them because they usually have tight connections with all community members.

3.5.2 Intracommunity File Searching and Retrieval

The query message is represented by a query vector: $v_Q = (t_0, w_0, t_1, w_1, t_2, w_2, \ldots, t_n, w_n)$. Each query is associated with a counter (count) indicating the number of hops it can travel. The count is decremented by one after each forward-ing. Since the query is initiated by users, term weights in v_Q are constant values. In the intracommunity searching, the destination that a query is sent to is represented by a combination of the v_Q and the node vector of the requester's community coordinator (v_{Nc}), represented by

$$v_{dest} = v_Q + (1 -)v_{Nc}: \quad (7)$$

In the first step, the requester calculates the similarity between the query vector and the community vector of the community it belongs to. If $sim(v_Q, v_C) < T_s$, the query is sent to the coordinator of the community directly (i.e., equals 0). Otherwise, equals 1 when the counter (count) is larger than 0 and 0 otherwise. This means that a requester first searches nearby nodes within count hops, and then resorts to its community coordinator. Specifically, the requester sends out a query to top F neighbors with the highest . Having > 1 copies of a request can enhance the efficiency of file searching. We call this strategy multicopy forwarding. To limit the number of copies for each request, we set $= \min_{i=0}^{k} F_i > g$, where F_i is the fitness of neighbor i and is the minimum delivery guarantee factor. The hop counter of a query is decreased by one after each forwarding. If the file is not found when count = 0, it is forwarded to the community coordinator (= 0). When a node receives multiple copies of query, it only processes the first one.

When node N_j receives a request, if $v_{dest} = v_Q$ and $sim(v_{dest}, v_{Nj})$ reaches the similarity threshold specified by the requester, it first tries to send the satisfied files back to the requester along the original path. If a forwarder on the path is not available due to node mobility, IRA is used to forward the file. Otherwise, N_j uses IRA to further forward the query. If $v_{dest} = v_{Nc}$ and N_j is not the coordinator N_c, N_j uses IRA to forward the request to N_c. After N_c receives the query, it checks its file indexes. If the indexes have files satisfying the request, the coordinator sends the request to the file holder when meeting it, which then sends the file back to the requester. Otherwise, N_c initiates the intercommunity file searching. Algorithm 1 shows the pseudocode of the intracommunity searching algorithm.

Algorithm 1. Pseudocode of intracommunity file searching for query Q conducted by node N_i.

```
Procedure intraSearchForQ ()
  if a neighbor nb of Ni matches query Q then
      Ni.sendQeuryTo(Q, nb)
  else if Q.src = Ni then
      if Sim(vQ, vC) < Ts then
          Q.vdest = vNC
          Ni.sendThroughIRATo(Q, NC )
      else
          Q.vdest = vQ
          Ni.rankNbByFitness()
          overallF = 0
          for each neighbor nb of node Ni do
              overallF = overallF + F(Q, nb)
              Ni.sendQeuryTo(Q, nb)
              if overallF > then
                  break
  else
      if Q.hops < MaxHop then
          Q.vdest = vQ
          Ni.rankNbByFitNess()
          nb = the neighbor with maximal fitness
          Ni.sendQeuryTo(Q, nb)
      else
          Q.vdest = vNC
          Ni.sendThroughIRATo(Q, NC )
```

3.5.3 Intercommunity File Searching and Retrieval

In the intercommunity searching algorithm, a coordinator maps a request to the foreign community that is most likely to contain the queried file. Similar to the intracommunity search step, the coordinator also uses the multicopy forwarding strategy, i.e., it sends out a query to

ambassadors having the highest similarity with the query to enhance the efficiency of the forwarding. We limit the number of copies for each request by letting $= \min_{i=0}^{k} sim(v_Q, v_C) > g$, where is the minimum delivery guarantee factor. Ambassadors then forward the request to corresponding foreign communities.

Upon receiving the request, the coordinator in the foreign community checks its file index to see if its community has the file. If not, the coordinator repeats the intercommunity file searching by looking up its ambassa-dors to check for further forwarding opportunities. If the file exists, the coordinator asks for the file from the file holder when meeting it and sends the file back to the requester's community through the corresponding ambas-sador. The coordinator of the requester's community will further forward the file to the requester.

Fig. 2 depicts the process of file searching, in which a requester (node R) in community C_1 generates a file request. Since its neighbors within count hops do not have the file, the request is then forwarded to the community coordinator N_{C1}. N_{C1} checks the community file indexes but still cannot find the file. It then asks the community

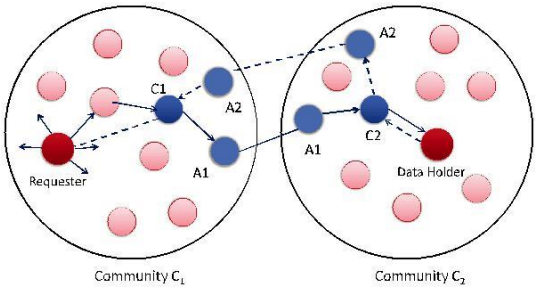


Fig. 2. File searching in SPOON.

ambassador N_{A1} to forward the request to the foreign community matching the queried file. Using the same way as N_{C1}, the community coordinator N_{C2} finds the file and sends it back to the requester's community via ambassador N_{A2}. The file is first sent to N_{C1}, and then forwarded to the requester. Algorithm 2 shows the pseudocode of the intercommunity searching algorithm.

Algorithm 2. Pseudocode of intercommunity file searching for query Q conducted by node N_i.

```
Procedure interSearchForQ ()
  if Ni is a coordinator then
    bContain      Ni.checkContainFile(Q)
    if bContain
        Ni. sendQeuryToDes(Q)
    else
        Ni. rankAmByMatch()
        overallS     0
        for each ambassador NA of Ni's community do
            n.sendQeuryTo(q, NA)
            overallS overallS þ Simðq:VQ; NA:VC Þ if
            overallS >
                break
if Ni is an ambassador then
    when Ni meets another node Nj
    if Nj. homeCommunity ¼ Ni. foreighCommunity then

        Ni. sendQeuryTo(Q, Nj)
        Nj. sendThroughIRATo(Q, NC )
```

3.6 Information Exchange among Nodes

We summarize the information exchanged among nodes in SPOON. In the community construction phase, two encountered nodes exchange their interest vectors and community vectors, if any, for community construction. In the role assignment phase, nodes broadcast their degree centrality within their communities for coordinator selec-tion. When the coordinator is selected, the coordinator ID is also broadcasted to all nodes in the community. Then, each node reports its contact frequencies with foreign communities to the coordinator for ambassador selection. Besides, when a node meets a coordinator of its commu-nity, the node also sends its updated node vector to the coordinator to update the community vector and retrieves the updated community vector from the coordinator. When an ambassador meets the coordinator of its community, it reports the community vectors of foreign communities to the coordinator. After above information exchange, two encountered nodes exchange their node vectors and history vectors for packet routing. Each node checks packets in it sequentially to decide which packets should be forwarded to the other node based on the file searching algorithm introduced in Section 3.5. Further, when network turns to be stable, the frequency of information exchange for community construction and node role assignment can be reduced to save costs.

3.7 Intelligent File Prefetching

Ambassadors in SPOON can meet nodes holding different files because they usually travel between different commu-nities frequently. Taking advantage of this feature, an ambassador can intelligently prefetch popular files outside of its home community. Recall that a query in a local community for a file residing in a remote community is forwarded through the coordinator of the local community. Thus, each coordinator keeps track of the frequency of local queries for remote files and provides the information of popular remote files to each ambassador in its community upon encountering it. When a community ambassador finds that its foreign community neighbors have popular remote files that are frequently requested by its home community members, it stores the files on its memory. The prefetched files can directly serve potential requests in the ambassador's home community, thus reducing the file searching delay.

3.8 Querying-Completion and Loop-Prevention

Given a file query, there may exist a number of matching files in the system. A node can associate a parameter S_{max} with its query to specify the number of files that it wishes to find. A challenge we need to handle is to ensure that the querying process stops when S_{max} matching files are discovered when multicopy forwarding is used. To solve this problem, we let a query carry S_{max} when it is generated. When a query finds a file that matches the query and is not discovered before, it decreases its S_{max} by one. Also, if this query is replicated to another node, S_{max} is evenly split to the two nodes. A query stops searching files

when its S_{max} equals 0.

When a query needs to find more than one file, it is likely that IRA would forward a query to the same node repeatedly. To avoid this phenomenon, SPOON incorpo-rates two strategies. First, the query holder inserts its ID to the query before forwarding the query to the next node. Second, a node records the queries it has received within a certain period of time. The former method avoids sending a packet to nodes it has visited before, while the latter method prevents sending different replicas of the same query to the same node. Specifically, when a node, say N_i, needs to forward a query to a newly met node N_j based on IRA, N_i checks whether the query's record of traversed nodes contains N_j. If yes, N_i does not forward the query to N_j. Also, when a node receives a query, if the query exists in its record of received queries, the node sends the query back to the sender. These two strategies effectively avoid searching loops by simply preventing a node from forwarding the same query to nodes that have received the query before.

3.9 Node Churn Consideration

In SPOON, when a node joins in the system, it first finds the communities it belongs to and learns the IDs of community coordinators, and then reports its files and utility values to the community coordinator when encountering it. This enables the coordinator to maintain updated information of the community members.

A node may leave the system voluntarily when users manually stop the SPOON application on their devices. In this case, a leaving node informs its community coordi-nator about its departure through IRA. If the leaving node is an ambassador, the coordinator then chooses a new ambassador. If the leaving node is a coordinator, it uses broadcast to notify other community members to select a new coordinator.

A node may also leave the system abruptly due to various reasons. Simply relying on the periodical beacon message, a node cannot tell whether a neighbor is left or is just isolated from itself, which is a usual case in MANETs. To handle this problem, each node records the time stamps when it meets other nodes, and sends it to the coordinator through IRA. The coordinator receives this information and updates the most recent time stamp of each node seen by other nodes. If the coordinator finds that a node's time stamp is more than T_x seconds ago, it considers this node as a departed node. Similarly, normal nodes in a community also maintain and update the time stamp of the coordinator to determine whether it is still alive. A node piggybacks the coordinator departure information on the beacon messages. Then, its nearby nodes can know whether the coordinator has left. Note that a node can know the number of community members from the coordinator. When a node finds that more than half of community members have found that the coordinator has left, it broadcasts a coordinator reelection message to select a new coordinator using the same method explained in Section 3.4.1.

4 Performance Evaluation

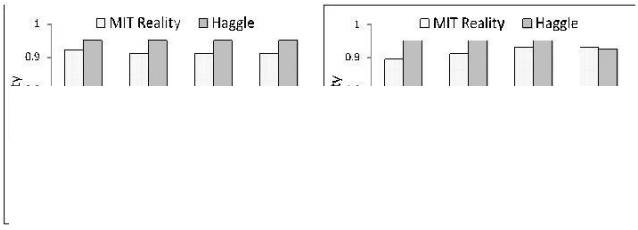
We evaluated the performance of SPOON in comparison with MOPS [18], PDIþDIS [8], [12], CacheDTN [14], PodNet [16], and Epidemic [34]. MOPS is a social network-based content service system. It forms nodes with frequent contacts into a community and selects nodes with frequent contacts with other communities as brokers for intercommunity communication. PDIþDIS is a combi-nation of PDI [8] and an advertisement-based DISsemina-tion method (DIS) [12]. PDI provides distributed search service through local broadcasting (three hops), and builds content tables in nodes along the response paths, while DIS let each node disseminate its contents to its neighbors to create content tables. CacheDTN replicate files to network centers in decreasing order of their overall popularity. In PodNet, nodes cache files interested by them and nodes they have met. We adopted the "Most Solicited" file solicitation strategy in PodNet. We doubled the memory on each node in CacheDTN and PodNet for replicas. In Epidemic, when two nodes meet each other, they exchange the messages the other has not seen. We have conducted the following experiments:

1. Evaluation of community construction. We first evaluated the proposed community construction algo-rithm introduced in Section 3.3.
2. GENI experiments. We deployed the systems on the real-world GENI ORBIT testbed [35], [36] and tested the performance using the MIT Reality trace. The GENI ORBIT testbed contains 400 nodes with 802.11 wireless cards. Nodes can communicate with each other through the wireless interface. We used real trace to simulate node mobility in ORBIT: two nodes can communicate with each other only during the period of time when they meet in the real trace.
3. Event-driven experiments with real trace. We also conducted event-driven experiments with two real traces.
4. Evaluation of enhancement strategies. We evaluated the effect of the enhancement strategies introduced in Sections 3.7, 3.8, and 3.9 through event-driven experiments.
5. NS2 experiments with synthetic mobility. We conducted experiments on NS-2 [37] using a community-based mobility model [38] to evaluate the applicability of SPOON in different types of networks. Due to page limit, the results are shown in Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2012.239.

We disabled the intelligent prefetching and the multi-copy forwarding (i.e., ¼ 1 and ¼ 1) in SPOON to make the method comparable. Also, because the comparison methods can return only one file for a query, we set S_{max} ¼ 1 in SPOON. In each community, we used the node having the most contacts with other communities as the ambassa-dor in SPOON and as the broker in MOPS. We also set the node with the most contacts with its community members as the coordinator in SPOON.

Besides the Haggle trace, we further tested with the MIT Reality trace [39], in which 94 smartphones were deployed among students and staffs at MIT to record their encounter-ing. The two traces last 0.34 million seconds (Ms) and 2.56 Ms, respectively. As in MOPS, we used 40 percent of the two traces to detect groups in which nodes share frequent contacts. Here, we use "group" to represent a group of nodes with frequent contacts, and use "commu-nity" to represent a group of nodes with common interests and frequent contacts. We got seven and eight groups for the MIT Reality trace and the Haggle trace, respectively. Then, because there is no real trace for P2P over MANETs, we collected articles from different news categories (e.g., sports, entertainment, and technology) from CNN.com and mapped them to the identified communities. Each node contains 50 articles from the news category for its community. Each node extracts its interests from its stored files. The similarity threshold was set to 70 percent in AGNES for file classification.

In experiments with the Haggle trace and the MIT Reality trace, we set the initialization period to 0.09 Ms and 0.3 Ms, the query generation period to 0.1 Ms and 1 Ms, and the TTL of a query to 0.15 Ms and 1.2 Ms, respectively.



(a) $h_2 = 30$ and $h_1 = 20 - 50$. (b) $h_1 = 30$ and $h_2 = 20 - 50$.

Fig. 3. Average similarity values with different h_1 and h_2.

Considering that people usually generate queries according to their interests, we set 70 percent of total queries searching for files located in the local community. Each query is for an article randomly selected from the article pool. We measured following metrics:

1. Hit rate. The percentage of requests that are success-fully delivered to the file holders.
2. Average delay. The average delay of the successfully delivered requests.
3. Maintenance cost. The total number of all messages except the requests, which are for routing information establishment and update, or replication creation.
4. Total cost. The total number of messages, including maintenance messages and requests, generated in a test.

4.1 Evaluation of Community Construction

We first tested the effectiveness of the community construc-tion method in SPOON, denoted by SPOON-CC, in comparison with Active-CC and Centralized-CC. The Active-CC selects three most active nodes to collect node contacts and interests when they meet nodes. In Centralized-CC, we purposely let a super node collect all node contacts and interests timely. Both Active-CC and Centralized-CC use AGNES to build communities with the collected information.

Since there is no real trace about content sharing in P2P MANETs, we tested in an indirect way. We first conducted the group construction and content distribution as previously described, and then removed the group identity of each node. Then, we run the three methods to create communities. After this, we matched each new community to the most similar old community and calculated the similarity value by $N_{sm}^2 = \delta N_p N_n \Phi$, where N_{sm} is the number of common nodes, N_p and N_n denote the size of the old community and the new community, respectively. In SPOON-CC, we set T_G to 1 to ensure interest closeness, and set h_1 and h_2 to 30. Active-CC and Centralized-CC used the same threshold for granting community membership as SPOON-CC. The average similarities in SPOON-CC, Active-CC, and Centralized-CC are 0.95, 0.87, and 1 with the Haggle trace and 0.91, 0.85, and 1 with the MIT trace, respectively. The Centralized-CC has inferior performance because active nodes can only collect information from nodes they have met, leading to less accurate community construction. Also, the perfor-mance of SPOON-CC is close to that of Centralized-CC, which has the best performance in theory. Such a result shows the effectiveness of SPOON-CC in this test.

TABLE 3
Efficiency and Cost in the Experiments on GENI

Method	Hit Rate	Ave. Delay (s)	Maintenance Cost	Total Cost
SPOON	0.671	152731.3	258764	275312
MOPS	0.629	163282.5	310131	320412
CacheDTN	0.5712	219021.4	283210	298123
PodNet	0.5932	183621	223218	240238
PDI+DIS	0.524	7418.9	298641	359841
Epidemic	0.8813	15621.2	669193	860475

We further varied h_1 and h_2 from 20 to 50 to verify the effectiveness of SPOON-CC. Figs. 3a and 3b show the average similarity values with the two traces. We see that the similarity values are above 90 percent with various h_1 and h_2. Such results further confirm the effectiveness of SPOON's community construction algorithm to cluster nodes with frequent contacts and similar interests in the experiments with the two real traces.

4.2 GENI Experiments

Table 3 shows the results of the GENI experiments of the six methods. From the table, we find that Epidemic generates the highest hit rate with the highest total cost and a low average delay. This is resulted from the dissemination nature of broadcasting. SPOON produces the second highest hit rate at the second lowest total cost and relatively high average delay. This is because SPOON utilizes both contact and content properties of social networks to guide file querying. Therefore, it can successfully locate queried files without the need of many information exchanges and request messages, though at a relatively slow speed. SPOON outperforms MOPS in terms of hit rate, delay, and cost. This is because SPOON utilizes IRA for intracommunication and dedicated ambassadors for inter-communication, while MOPS relies heavily on brokers. Also, MOPS only considers node contact in routing, while SPOON considers both content and contact. We will elaborate the reasons in describing the trace driven simulation results later on.

CacheDTN has low hit rate, median cost, and high average delay. This is because though replicas are created, queries wait for files passively on their originators, leading to a long delay. Also, the replication of files to network centers incurs a high cost. PodNet has low hit rate for the same reason as CacheDTN. However, because the replicas on each node are more catered for the interests of itself and nodes it has met, PodNet has slightly higher hit rate than CacheDTN. Moreover, PodNet has the lowest cost because nodes in it only replica files they are interested in. PDIþDIS generates the lowest hit rate at relatively high total cost and low average delay. The low hit rate is caused by the poor mobility resilience of route tables. As a result, only partial queries are resolved quickly in the local broadcasting. Others passively wait for file holders or updated routes and usually cannot be resolved timely. Then, because we only count the average delay of successful queries, PDIþDIS has the lowest average delay.

We also evaluated the performance of each method in memory utilization in terms of the average number of queries in the buffer (Query) and the average size of a content/ neighbor table (Table). The results are shown in Table 4. For

TABLE 4
Memory Usage in the Experiments on GENI

Metric	SPOON	MOPS	CacheDTN	PodNet	PDI+DIS	Epid.
Query	36.8	44.1	48.4	45.3	13.1	1998
Table	10.4	16.9	50	50	15.6	0

the average number of buffered queries, we find that PDIþDIS<SPOON<MOPS<PodNet<CacheDTN<Epidemic. In Epidemic, nodes buffer the most queries because it tries to replicate each request to all nodes in the system. CacheDTN and PodNet have a lot of queries in memory because they do not actively search for the queried files. Both SPOON and MOPS keep one copy of each query during the searching process. However, because SPOON completes file query more quickly than MOPS (as shown in Table 3), it buffers fewer quires in memory than MOPS. PDIþDIS has the fewest number of queries on nodes because local broadcasting just forwards queries without buffering.

Considering that each entry in the content table has roughly the same size, we used the number of entries in a table to represent memory usage. The results in Table 4 show that Epidemic<SPOON<MOPS<PDIþDIS< CacheDTN¼PodNet. Clearly, Epidemic does not need memory on content table. SPOON stores the second fewest content synopses because most nodes only store the information of the same community members. In MOPS, brokers store content synopses of all nodes in their communities and consume a large amount of memory. Therefore, MOPS produces high average num-ber of stored content synopses. PDIþDIS stores a large amount of content synopses because each node collects content synopses from all nodes it has met and from all received reply messages. CacheDTN and PodNet have the highest number of entries in the content table because we doubled the memory (i.e., 50 articles) for replicas. In summary, the results in Tables 3 and 4 show that SPOON is superior over other methods in terms of hit rate, average delay, total cost, and memory efficiency.

4.3 Event-Driven Experiments with Real Trace

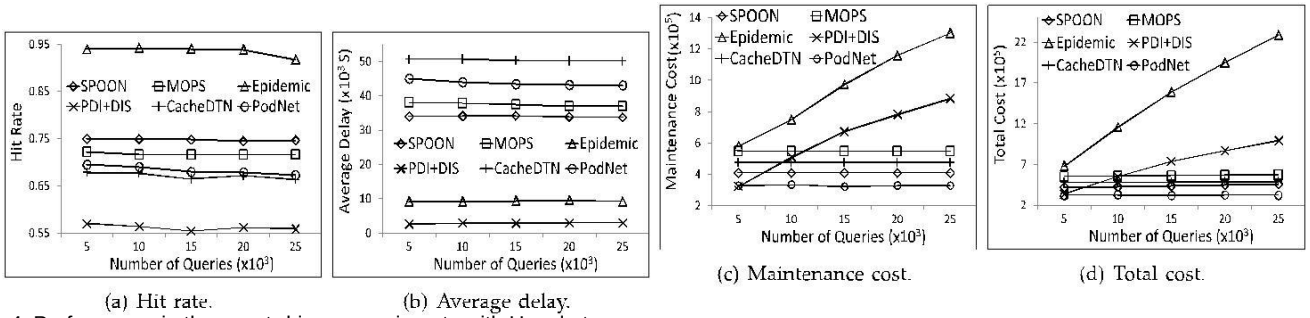
In this experiments, we varied the total number of queries from 5,000 to 25,000 to show the scalability of each method in terms of the amount of queries.

4.3.1 Hit Rate

Figs. 4a and 5a show the hit rate of each method in the experiments with the Haggle trace and the MIT Reality trace, respectively. We find that with both traces, Epidemic can resolve almost all requests, while PDIþDIS can only complete about 60 percent of requests. The hit rates of SPOON, MOPS, PodNet, and CacheDTN reach about 75, 70, 68, and 67 percent, respectively. Epidemic has the highest hit rate because of its broadcasting nature. In SPOON, coordinators and ambassadors facilitate intra- and inter-community searching, while the IRA actively forwarded to the node with a high probability of meeting the destination. MOPS only relies on the encountering of mobile brokers for file searching. This probability is lower than that of SPOON, resulting in a lower hit rate. PodNet and CacheDTN lack active query forwarding, leading to median hit rates. However, replicas on each node are more catered to the interests of nodes it can meet in PodNet, while CacheDTN just caches files on network center. Therefore, PodNet has slightly higher hit rate than CacheDTN. In PDIþDIS, many routes in the content table expire quickly due to node mobility. As a result, most successful requests are resolved through the 3-hop broadcast. Others have to passively wait for file holders or updated routes. Therefore, many requests cannot be resolved, leading to a low hit rate.

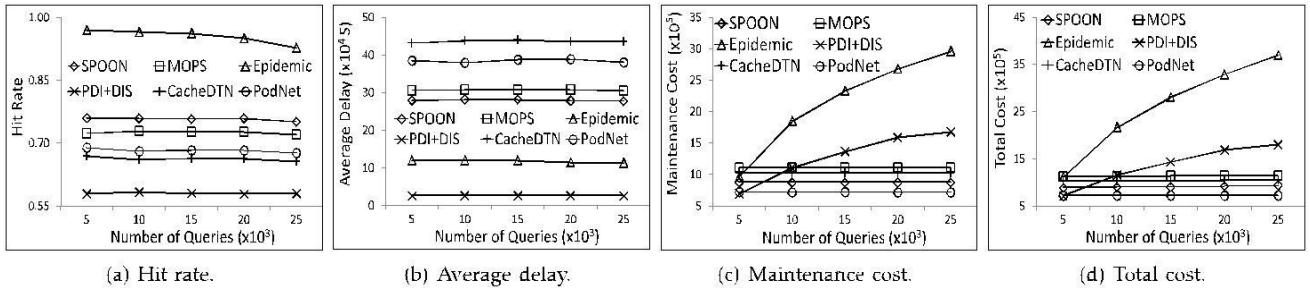
4.3.2 Average Delay

Figs. 4b and 5b show the average delays of the six methods in the tests with the Haggle trace and the MIT Reality



(a) Hit rate. (b) Average delay. (c) Maintenance cost. (d) Total cost.

Fig. 4. Performance in the event-driven experiments with Haggle trace.



(a) Hit rate. (b) Average delay. (c) Maintenance cost. (d) Total cost.

Fig. 5. Performance in the event-driven experiments with MIT Reality trace.

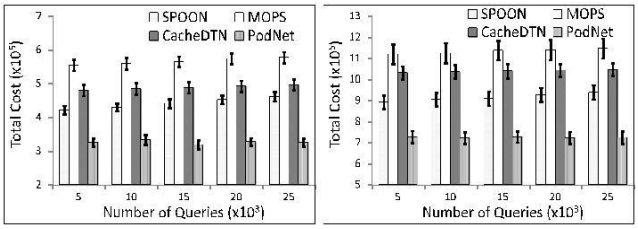


Fig. 6. Total costs with confidence intervals.

trace, respectively. The delays follow PDIþDIS<Epidemic< SPOON<MOPS<PodNet<CacheDTN.

Recall that we only measure the delay of successful requests. In PDIþDIS, most successful requests are resolved in the initial 3-hop broadcasting stage. Therefore, it generates the least average delay. In Epidemic, requests are rapidly distributed to nodes at the cost of multiple copies. As a result, requests can reach their destinations quickly. MOPS exhibits a large delay because requests in it usually have to wait for a long time for brokers or same-community file holders. In contrast, SPOON always tries to find an optimal neighbor to send a request to the file holder with the interest-oriented routing algorithm. In addition, the designation of ambassadors in SPOON increases the possibility of relaying requests to foreign communities. As a result, SPOON has lower average query delay than MOPS. PodNet and CacheDTN generate high average delay because queries only wait for file holders passively on their originators. However, because PodNet create replicas that are more likely to be encountered by nodes that are interested in them, it has lower average delay than CacheDTN.

4.3.3 Cost

Figs. 4c and 5c plot the maintenance costs of the six methods in the experiments with the Haggle trace and the MIT Reality trace, respectively. We see that when the total number of queries is small, the six methods all have low maintenance cost. When the total number of queries is larger than 10,000, the maintenance costs generally follow: Epidemic>PDIþDIS>MOPS>CacheDTN>SPOON>PodNet.

In PodNet, nodes only replica interested files when meet other nodes, leading to the least maintenance cost. In SPOON, nodes exchange node vectors for the update of history vector. Nodes also report its contents to coordina-tors for file indexing. In MOPS, brokers exchange the contents of all nodes from their home communities when meeting each other. Therefore, MOPS produces slightly higher cost than SPOON. The active replication of files to network centers in CacheDTN leads to a high cost. PDIþDIS needs to build content tables through reply messages and disseminated queries, so it has higher maintenance cost than above four methods when the number of queries becomes large. In Epidemic, two nodes need to inform each other requests already on them, which causes a lot of information exchange and leads to the highest maintenance cost.

We see that when the number of queries increases, the maintenance costs of SPOON, PodNet, CacheDTN, and MOPS remain stable while those of Epidemic and PDIþDIS increase quickly. This is because the maintenance costs of the former four methods are determined by the information/replication exchanges among nodes and are irrelevant with the number of queries, and those of Epidemic and PDIþDIS are related to the total number of queries. Such results prove the scalability of SPOON, MOPS, CacheDTN, and PodNet in query amount.

Figs. 4d and 5d show the total cost of each method in the experiments with the Haggle trace and the MIT Reality trace, respectively. In the two figures, the results of MOPS, CacheDTN, SPOON, and PodNet are shown to be very close. We then plot the total costs of the four methods with 95 percent confidence interval in Figs. 6a and 6b for better demonstration. Note we did not show the confidence interval of other measurements because they have clear difference and the page limit. We find that the total costs follow Epidemic>PDIþDIS>MOPS>CacheDTN>SPOON>PodNet, which is the same as Figs. 4c and 5c. Such a result means that the maintenance cost is the majority part of the total cost. With above results, we conclude that SPOON has the highest overall file searching efficiency in terms of hit rate, delay, and cost.

TABLE 5
Hit Rate Improvement with the Haggle Trace

# of packets	Original	Prefetching	Multi-copy forwarding
5000	0.75137	0.75313	0.779412
10000	0.75215	0.75633	0.780135
15000	0.73831	0.74274	0.778912
20000	0.74928	0.75242	0.774321
25000	0.74731	0.75201	0.779415

4.4 Evaluation of the Enhancement Strategies

4.4.1 Multicopy Forwarding and Prefetching

We first evaluated the effect of the multicopy forwarding and the intelligent file prefetching. We let "Multicopy forwarding" and "Prefetching" denote the SPOON with the corresponding improvement strategy, respectively, and compare them with the "Original" SPOON. In multicopy forwarding, we let each query originator distribute two copies of its query. In Prefetching, we let each ambassador store top 10 most popular files. We varied the number of queries from 5,000 to 25,000. The test results are shown in Tables 5 and 6.

We find that the multicopy forwarding strategy with only two copies enhances the hit rate greatly in the experiments with both traces. This is because when each query has two copies in the system, its probability of encountering the node containing the queried file increases. Such a result shows the effectiveness of the multiforwarding strategy.

TABLE 6
Hit Rate Improvement with the MIT Reality Trace

# of Packets	Original	Prefetching	Multi-copy forwarding
5000	0.761371	0.7671675	0.780413
10000	0.759941	0.762841	0.770838
15000	0.760135	0.763762	0.772843
20000	0.756418	0.759957	0.773901
25000	0.751835	0.754837	0.771963

TABLE 7
Effect of Query-Completion Strategy

Trace	SPOON-OR	SPOON-QC	SPOON-QCLP
Hit Rate			
Haggle	0.8741	0.8701	0.8813
MIT Reality	0.9091	0.9012	0.9406
Number of query forwarding operations			
Haggle	569841	530516	304953
MIT Reality	721852	609863	249132

We also see from the two tables that the file prefetching strategy slightly improves the hit rate with both the two traces. This is because 1) we only configure two ambassa-dors per community, and 2) the prefetched files only satisfy a small amount of queries because most queries are for contents in local community. The improvement on the hit rate still demonstrates the effectiveness of the file prefetch-ing strategy and the improvement would be greater with more ambassadors and greatly varied file popularity.

4.4.2 Querying-Completion and Loop-Prevention

We name SPOON without querying-completion as "SPOON-OR", SPOON with the querying-completion only as "SPOON-QC", and SPOON with both querying-comple-tion and loop-prevention as "SPOON-QCLP". We set the number of queries to 15,000. In SPOON-OR, queries do not stop searching until TTL expiration. We also set the maximal number of files each query can retrieve, S_{max}, to

2. To enable a query to find two files, we purposely let each file have two copies in the system. To alleviate the influence of the TTL on the evaluation of the querying-completion, we enlarge the TTL to the entire length of the used traces. The test results are shown in Table 7.

We find from the table that SPOON-QCLP has slightly higher hit rate than SPOON-QR and SPOON-QC. This is because the loop-prevention avoids forwarding a query to the same file holder repeatedly, thereby utilizing forward-ing opportunities more efficiently. SPOON-QC has slightly

lower hit rate than SPOON-OR because it stops querying when S_{max} files are fetched. We also see that the number of query forwarding operations follow SPOON-OR>SPOON-QC>SPOON-QCLP. This is because SPOON-OR does not stop querying until the TTL is expired. SPOON-QC reduces the cost as it stops querying after the specified number of files are located. In SPOON-QCLP, the loop-prevention avoids redundant forwarding to the same node, leading to less number of forwarding operations and more efficient file searching.

TABLE 8
Effect of the Detection of Coordinator Departures

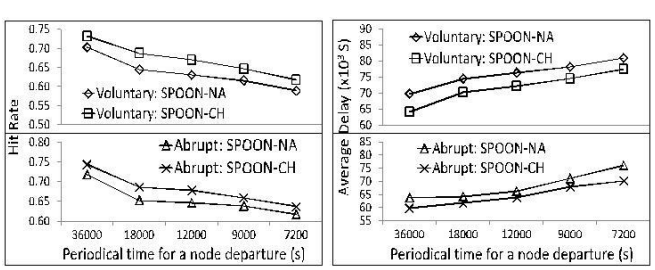
Trace	w/o churn consideration	w/ churn consideration-Ab	w/ churn consideration-Vo
Haggle	0.650643	0.684512	0.729942
MIT Reality	0.641053	0.677841	0.746841

4.4.3 Node Churn Consideration

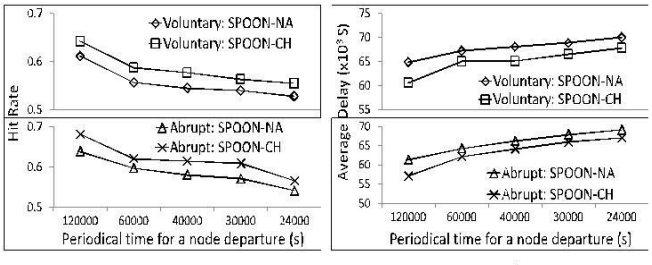
We name SPOON with and without a strategy to handle node churn as "SPOON-CH" and "SPOON-NA," respec-tively. The total number of queries was set to 15,000. The period for beacon message was set to 100 and 1,000 s with the Haggle trace and the MIT Reality trace, respectively. In the test, N_L nodes leave the system evenly during the first 1/2 and 1/4 of the Haggle trace and the MIT Reality trace, respectively. N_L was varied from five to 25. We name the node that contains a file matching a query as the primary matching node for the query. To demonstrate the perfor-mance of SPOON in node churn, for each queried file, we purposely created a file that has 70 percent similarity with it in a nonleaving node in the same community with the primary matching node. We name this node as the secondary matching node.

The test results of voluntary and abrupt normal nodes departure are shown in Fig. 7. We see that in all cases, when node churn consideration is applied, the hit rate is increased and the average delay is decreased. This is because with the node churn consideration, queries failing to find their primary matching nodes (i.e., have left the system) are further forwarded to their secondary matching nodes while when there is no node churn consideration, these queries just wait on coordinators for the primary matching node, leading to a low hit rate and a high average delay. We also observe that when the number of leaving nodes increases, the hit rate decreases and the average delay increases. This is because leaving nodes can no longer relay queries or departure notification/detection messages, leading to lower hit rate and higher average delay.

We also tested the scenario in which only the coordi-nator nodes leave the system. Since the total number of coordinators is limited, we only randomly chose two coordinators to leave the system during the test. We name the scenarios when coordinators depart abruptly and voluntarily as "w/churn consideration-Ab" and "w/churn consideration-Vo," respectively. The test results are shown in Table 8. We find that when coordinators leave the



(a) Hit rate with Haggle trace. (b) Ave. delay with Haggle trace.



(c) Hit rate with MIT trace. (d) Ave. delay with MIT trace.

Fig. 7. Performance with voluntary and abrupt node departures.

system, the hit rate of SPOON with node churn considera-tion is much higher than that without node churn consideration. This is because the coordinator is critical in both intra- and interfile searching in SPOON. Without node churn consideration, queries just wait for coordinators until their TTL expiration if they need to be forwarded to coordinators, leading to a low hit rate and a high average delay. Above results show that SPOON's strategies for node churn consideration can improve the system perfor-mance at a low cost.

5 Conclusion

In this paper, we propose a social network-based P2P cOntent file sharing system in disconnected mObile ad hoc Networks. SPOON considers both node interest and contact frequency for efficient file sharing. We introduce four main components of SPOON: Interest extraction identifies nodes' interests; Community construc-

tion builds common-interest nodes with frequent contacts into communities. The node role assignment component exploits nodes with tight connection with community members for intracommunity file searching and highly mobile nodes that visit external communities frequently for intercommunity file searching; The interest-oriented file searching scheme selects forwarding nodes for queries based on interest similarities. SPOON also incorporates addi-tional strategies for file prefetching, querying-completion, and loop-prevention, and node churn consideration to further enhance file searching efficiency. The system deployment on the real-world GENI Orbit platform and the trace-driven experiments prove the efficiency of SPOON. In future, we will explore how to determine appropriate thresholds in SPOON, how they affect the file sharing efficiency, and how to adapt SPOON to larger and more disconnected networks.

References

[1] "The State of the Smartphone Market," http://www.allaboutsymbian.com/news/item/6671_The_State_of_the_Smartphone_ Ma.php, 2013.

[2] "Next Generation Smartphones Players, Opportunities & Fore-casts 2008-2013," technical report, Juniper Research, 2009.

[3] "A Market Overview and Introduction to GyPSii," http://corporate.gypsii.com/docs/MarketOverview, 2013.

[4] "Bittorrent," http://www.bittorrent.com, 2013.

[5] "Kazaa," http://www.kazaa.com, 2013.

[6] M. Papadopouli and H. Schulzrinne, "A Performance Analysis of 7DS: A Peer-to-Peer Data Dissemination and Prefetching Tool for Mobile Users," Proc. IEEE Sarnoff Symp. Digest Advances in Wired and Wireless Comm., 2001.

[7] A. Klemm, C. Lindemann, and O. Waldhorst, "A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks," Proc. IEEE 58th Vehicular Technology Conf. (VTC '03), 2003.

[8] C. Lindemann and O.P. Waldhort, "A Distributed Search Service for Peer-to-Peer File Sharing," Proc. Int'l Conf. Peer-to-Peer Computing (P2P '02), 2002.

[9] D.W.A. Hayes, "Peer-to-Peer Information Sharing in a Mobile Ad Hoc Environment," Proc. IEEE Sixth Workshop Mobile Comput-ing Systems and Applications (WMCSA '04), 2004.

[10] J.B. Tchakarov and N.H. Vaidya, "Efficient Content Location in Wireless Ad Hoc Networks," Proc. IEEE Int'l Conf. Mobile Data Management (MDM '04), 2004.

[11] C. Hoh and R. Hwang, "P2P File Sharing System over MANET based on Swarm Intelligence: A Cross-Layer Design," Proc. IEEE Wireless Comm. and Networking Conf. (WCNC '07), pp. 2674-2679, 2007.

[12] T. Repantis and V. Kalogeraki, "Data Dissemination in Mobile Peer-to-Peer Networks," Proc. Sixth Int'l Conf. Mobile Data Management (MDM '05), 2005.

[13] Y. Huang, Y. Gao, K. Nahrstedt, and W. He, "Optimizing File Retrieval in Delay-Tolerant Content Distribution Community," Proc. IEEE 29th Int'l Conf. Distributed Computing Systems (ICDCS '09), 2009.

[14] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa, "Supporting Cooperative Caching in Disruption Tolerant Networks," Proc. 31st Int'l Conf. Distributed Computing Systems (ICDCS '11), 2011.

[15] J. Reich and A. Chaintreau, "The Age of Impatience: Optimal Replication Schemes for Opportunistic Networks," Proc. Fifth Int'l Conf. Emerging Networking Experiments and Technologies (CoNEXT '09), 2009.

[16] V. Lenders, M. May, G. Karlsson, and C. Wacha, "Wireless Ad Hoc Podcasting," ACM SIGMOBILE Mobile Computing and Comm. Rev., vol. 12, pp. 65-67, 2008.

[17] K. Chen and H. Shen, "Global Optimization of File Availability through Replication for Efficient File Sharing in MANETs," Proc. IEEE 19th Int'l Conf. Network Protocols (ICNP), 2011.

[18] F. Li and J. Wu, "MOPS: Providing Content-Based Service in Disruption-Tolerant Networks," Proc. IEEE 29th Int'l Conf. Distributed Computing Systems (ICDCS '09), 2009.

[19] P. Costa, C. Mascolo, M. Musolesi, and G.P. Picco, "Socially-Aware Routing for Publish-Subscribe in Delay-Tolerant Mobile Ad Hoc Networks," IEEE J. Selected Areas in Comm., vol. 26, no. 5, pp. 748-760, June 2008.

[20] E. Yoneki, P. Hui, S. Chan, and J. Crowcroft, "A Socio-Aware Overlay for Publish/Subscribe Communication in Delay Tolerant Networks," Proc. 10th ACM Symp. Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM '07), 2007.

[21] C. Boldrini, M. Conti, and A. Passarella, "ContentPlace: Social-Aware Data Dissemination in Opportunistic Networks," Proc. 11th Int'l Symp. Modeling, Analysis and Simulation Wireless and Mobile Systems (MSWiM '08), 2008.

[22] A. Fast, D. Jensen, and B.N. Levine, "Creating Social Networks to Improve Peer-to-Peer Networking," Proc. 11th ACM SIGKDD Int'l Conf. Knowledge Discovery in Data Mining (KDD '05), 2005.

[23] A. Iamnitchi, M. Ripeanu, and I.T. Foster, "Small-World File-Sharing Communities," Proc. IEEE INFOCOM, 2004.

[24] M. Mcpherson, "Birds of a Feather: Homophily in Social Net-works," Ann. Rev. Sociology, vol. 27, no. 1, pp. 415-444 , 2001.

[25] E. Yoneki, P. Hui, S. Chan, and J. Crowcroft, "A Socio-Aware Overlay for Publish/Subscribe Communication in Delay Tolerant Networks," Proc. 10th ACM Symp. Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM '07), 2007.

[26] A. Chaintreau, P. Hui, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Impact of Human Mobility on Opportunistic Forwarding Algorithms," IEEE Trans. Mobile Computing, vol. 6, no. 6, pp. 606-620, June 2007.

[27] V. Carchiolo, M. Malgeri, G. Mangioni, and V. Nicosia, "An Adaptive Overlay Network Inspired by Social Behavior," J. Parallel and Distributed Computing, vol. 70, pp. 282-295, 2010.

[28] A. Iamnitchi, M. Ripeanu, E. Santos-Neto, and I. Foster, "The Small World of File Sharing," IEEE Trans. Parallel and Distributed Systems, vol. 22, no. 7, pp. 1120-1134, July 2011.

[29] H. Schu¨tze and C. Silverstein, "Projections for Efficient Document Clustering," Proc. 20th Ann. Int'l ACM Conf. Research and Development in Information Retrieval (SIGIR '07), pp. 74-81, 1997.

[30] P. Bonacich, "Factoring and Weighting Approaches to Status Scores and Clique Identification," J. Math. Sociology, vol. 2, pp. 113-120, 1972.

[31] L. Kaufman and P. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley and Sons, 1990.

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

[33] W. Hsu, T. Spyropoulos, K. Psounis, and A. Helmy, "Modeling Time-Variant User Mobility in Wireless Mobile Networks," Proc. IEEE INFOCOM, 2007.

[34] A. Vahdat and D. Becker, "Epidemic Routing for Partially-Connected Ad Hoc Networks," technical report, Duke Univ., 2000.

[35] "GENI Project," http://www.geni.net, 2013.

[36] "Orbit," http://www.orbit-lab.org, 2013.

[37] "The Network Simulator ns-2," http://www.isi.edu/nsnam/ns, 2013.

[38] M. Musolesi and C. Mascolo, "Designing Mobility Models Based on Social Network Theory," ACM SIGMOBILE Computing and Comm. Rev., vol. 11, pp. 59-70, 2007.

[39] N. Eagle, A. Pentland, and D. Lazer, "Inferring Social Network Structure Using Mobile Phone Data," Proc. Nat'l Academy of Sciences USA, vol. 106, no. 36, pp. 15274-15278, 2009.
[40] K. Chen and H. Shen, "Leveraging Social Networks for P2P Content-Based File Sharing in Mobile Ad Hoc Networks," Proc. IEEE Eighth Int'l Conf. Mobile Adhoc and Sensor Systems (MASS),2011.