This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems

Efficient File Search in Delay Tolerant Networks with Social Content and Contact Awareness

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

Abstract—Distributed file searching in delay tolerant networks formed by mobile devices can potentially support various useful applications. In such networks, nodes often present certain social network properties of their holders in terms of contents (i.e., interests) and contacts. However, current methods in DTNs only consider either content or contact for file searching or dissemination, which limits the file sharing efficiency. In this paper, we first analyze real traces to confirm the importance and necessity of considering both content and contact in file search. We then propose Cont^2, a social-aware file search method that exploits both node contents and contact patterns. First, considering people with common interests tend to share files and gather together, Cont^2 virtually groups common-interest nodes into a community to direct file search. Second, considering human mobility follows a certain pattern, Cont^2 exploits nodes' contact frequencies with a community to expedite file searching. To further improve the searching efficiency, Cont^2 also integrates sub-communities and parallel forwarding as optional components for file searching. Trace-driven experiments on the GENI testbed and NS-2 simulator show that Cont^2 can effectively improve the search efficiency compared to current methods.

Index Terms—Social-Aware, File Search, Delay Tolerant Networks

1 Introduction

The wide usage of portable digital devices (e.g., laptops and smart phones) has stimulated significant researches on distributed file search in mobile environments. In this paper, we envision DTNs as a backup network for infrastructure intensive areas or a low-cost communication structure in severe environments, e.g., mountain/rural areas and battle field. For example, even with no network connection, students can acquire course materials from other students' mobile devices [1] and drivers can acquire weather and traffic conditions from passing by vehicles [2]. Besides, people or vehicles moving in mountain areas can help forward data, e.g., emails, between villages at a very low cost, i.e., without the need of infrastructures [3]. Thus, in this paper, we focus on distributed peer-to-peer file search in a delay tolerant network (DTN) [4] formed by mobile devices, the holders of which exhibit certain social network properties.

However, due to sparse node distribution and continuous node mobility, DTNs are featured by frequent network partition and intermittent connections. As a result, packet forwarding is often realized in a store-carry-forward manner in DTN routing algorithms [3], [5]–[7], which means that a message is carried by current holder until meeting another forwarder. Furthermore, due to the distributed network structure, it is almost impossible to maintain global file distribution information in DTNs. This means that a file request often does not know which nodes contain the requests file when it is generated. These characteristics lead to significant challenges on efficient file searching in DTNs.

Kang Chen is with the Department of Electrical and Computer Engineering, Southern Illinois University, Carbondale, IL 62901 USA email: kchen@engr.siu.edu

Haiying Shen and Li Yan are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29631 USA e-mail: {shenh,lyan}@clemson.edu.

Recently, some methods [1], [8]–[16] have been proposed to leverage node contacts/interests for content dissemination or publish in DTNs. They either group nodes with frequent contact or forward contents following node interests for file service in DTNs. However, only considering interest/content or contact may lead to a low file searching efficiency. First, a node usually has multiple interests, and few nodes share many interests. This implies that contact based communities may hold files from different interests, leading to frequent inter-community search. Second, same interest nodes may not always stay together due to node mobility. Therefore, purely relying on node interests for file searching may not be able to find the file holder quickly. Our study on crawled Facebook data and a real trace obtained from students on a campus [17] confirms these reasons, as discussed later in Section 3.

Furthermore, file searching is different from data dissemination in two aspects. First, they have different directions. The former forwards a request to the content holder, while the latter distributes contents to interested nodes. Second, a request only servers one node, while a data can repetitively satisfy many nodes. Therefore, it would be desired to not replica a request to control the overhead, while a data may be replicated multiple times.

To overcome these shortcomings, we propose a social-aware Content and Contact based file search method, namely Cont^2, for DTNs in which the holders of mobile nodes present certain social network properties. Cont^2 is a single-copy file searching algorithm that utilizes both node contact and node interest to efficiently locate the requested file in DTNs. The cornerstone for the design of Cont^2 originates from two properties of the social networks in the DTN scenario:

(P1) Common interest: every node has social interests, and nodes with a common interest, though may be separated, tend to meet more often with each other than

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems

with other nodes [18];
(P2) Movement pattern: people usually present skewed visiting preferences to certain places [19].

By leveraging P1, we develop a social community creation algorithm that virtually classifies nodes into different communities based on their interests. This helps to forward a file request toward the destination community that contains the file. By leveraging P2, we develop a file searching algorithm that always forwards requests to nodes that are more likely to meet the requests content. On the basis of basic components, we also propose advanced components that exploit contact-based sub-communities and parallel forwarding to further enhance file searching efficiency. As a result, both content (i.e., interests) and contact are exploited to find appropriate forwarders for file requests, leading to a high file searching efficiency. In a nutshell, the major contribution of Cont^2 is to synergistically exploit both node contents (interests) and contacts under the social network context for efficient file searching in DTNs, while previous methods only utilize one of the two properties for content dissemination or publish service in DTNs.

The remainder of this paper is arranged as follows. Related works are described in Section 2. Section 3 introduces the analysis on real traces. Section 4 presents the detailed design of Cont^2. In Section 5, the performance of Cont^2 is evaluated through experiments. Finally, Section 6 concludes the paper with remarks on future work.

2 Related Work

2.1 Packet Routing in DTNs

DTNFLOW [3] uses node transit patterns between different landmarks to find the fastest path for efficient packet routing among landmarks in DTNs. RAPID [5] regards opportunistic encountering among nodes as resources in the network and converts packet routing into a resource allocation problem. The work in [6] exploits nodes' frequently visited communities to realize efficient multi-copy packet routing in mobile social networks. SMART [7] uses the social map that shows the social closeness between surrounding nodes to guide packet routing. However, packet routing algorithms in DTNs cannot be employed for file searching/sharing. This is because in DTNs, the destination node of a file request is not determined when it is generated.

2.2 Content Dissemination in DTNs

In Haggle project [1], data is forwarded along nodes with matched interests. MOPS [8] groups nodes with frequent contact into a community and selects nodes that frequently visit a neighboring community as brokers for inter-community communication. The socio-aware overlay [9] builds brokers into an overlay, in which brokers use unicast or direct communication protocols (e.g., WiFi) for communication. In the work of [10], the author considers users' impatience in acquiring files for optimal file caching in opportunistic networks. Lenders *et al.* [11] discussed different content solicitation strategies in the podcasting through the peer-to-peer communication among mobile devices. Zhang *et al.* [12] defined friends as nodes with similar interests and evaluated four data diffusion strategies. In Content-Place [13], nodes collect files that are possibly interested by nodes in their social communities. In addition to node interests, Gao *et al.* [14] further considered social contact patterns in data dissemination. This method forwards data to nodes that are more likely to meet nodes that are interested in the data. The works in [15] and [16] exploit transient node contacts and community structures to find key relay nodes for effective data dissemination.

These methods differ from Cont^2 in two aspects. First, they only utilize either contact or interest for data forwarding, while Cont^2 considers both properties. Second, these works mainly investigate data dissemination, which is different from the goal of data searching. Data dissemination distributes contents to interested users, while data searching forwards requests to file holders.

2.3 Content Searching in DTNs

There are also works on data searching/sharing in DTNs [20]–[23]. The works in [20] and [21] allow each node to delegate its file requests to other nodes to reach the requests files. However, a node simply chooses the node that it can meet frequently as its file request delegates, leading to a low file sharing efficiency.

The work in [22] investigates how to stop the content searching in file searching in DTNs so that the number of discovered files and searching overhead can be balanced. SPOON in our previous work [23] was developed for disconnected MANETs that have strong node interest and contact correlation, while Cont^2 is developed for DTNs where nodes with similar interests do not necessarily always stay together. Therefore, SPOON and Cont^2 have different file search algorithms. SPOON relies on the meeting frequency with a content to search for the content within an interest community, while Cont^2 utilizes intra-community mobility pattern to search for files. Cont^2 novelly builds *community contact table* and *neighbor table* to direct the file searching.

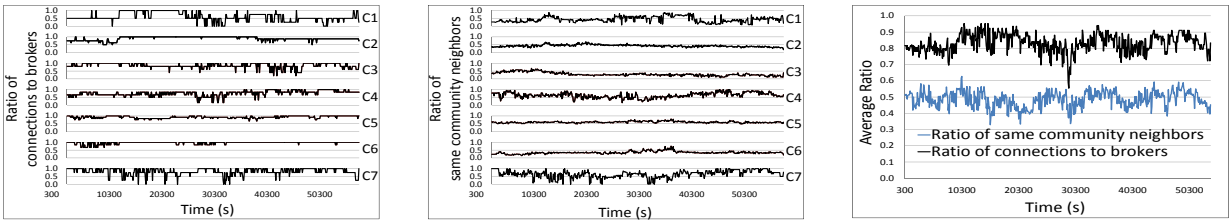
3 Real-Trace Analysis

3.1 MIT Reality Trace Analysis

We first analyzed the MIT Reality trace [17], which records the encountering of 94 smart phones held by students and staffs at MIT, to verify the drawbacks of only considering node contacts in file search in DTNs. Such a trace shows the typical scenario for the proposed Cont^2, i.e., campus, and is representative in analysis.

Using the method in MOPS [8], we classified nodes into 7 communities (C_{1-7}), and nodes in each community share frequent contacts. We also selected one pair of brokers for each community pair. A community's broker for another community, say C_x, is the node in the community that has the highest overall contact frequency

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems



(a) Ratio of connections to brokers. (b) Ratio of same community neighbors. (c) Overall average ratio.

Fig. 1. Real-trace analytical results.

with nodes in C_x. The overall contact frequency with a community is the sum of contact frequencies with nodes in the community. Due to page limit, we only show the analysis result for one day (15 hours=54000s). The results on other days show similar trend.

3.1.1 Connectivity to Brokers

A community's *ratio of connections to brokers* is the portion of community members that connect to at least one broker. Figure 1(a) plots this metric for each community every 100s and shows that the ratio varies in range [0.5,1]. Figure 1(c) plots the average value of all communities, which shows that the average value is around 0.8. This means that averagely, 20% of nodes cannot communicate with their brokers, and in some communities, 50% of nodes cannot communicate with their brokers.

O1: *Due to node mobility, community nodes do not always connect to their brokers.*

3.1.2 Connectivity to Community Members

A node's *ratio of same community neighbors* is the portion of its neighbors that belong to the node's community. Figure 1(b) plots the average of the ratios of all nodes in each community every 100s, and shows that the ratio varies greatly in [0.1, 0.9]. Figure 1(c) plots the average value of all communities and shows that the average value is around 0.5, which means that on average half of a node's neighbors are not from the same community.

O2: *Due to node mobility, nodes in one community do not always stay together.*

3.2 Facebook Data Analysis

3.2.1 Do nodes with a common interest also share many other interests?

We crawled the data of 117 users from Facebook [24] and examined their interest closeness. These people watched a video of a randomly selected user in Facebook. We identified 20 interests (e.g., sports, gaming and pop music) and found 86 users who have at least one of these interests. We found 1462 pairs of users that share at least one common interest. We then calculated the interest closeness between user i and j (c_{ij}) by: $c_{ij} = m_{ij}^2/s_i s_j$, where m_{ij} is the number of shared interests of the two users, and s_i and s_j are the number of interests of user i and user j, respectively.

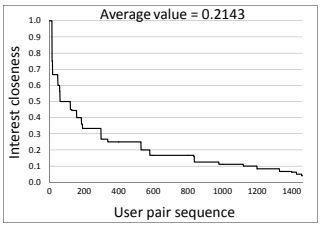


Fig. 2. Interest closeness of common-interest user pairs.

Figure 2 shows the interest closeness of different pairs of users. We observe that the interest closeness mainly varies in the range [0.1, 0.33] and few exceed 0.4, and the average closeness is only 0.2143. This result proves that though each pair of nodes shares at least one common interest, they rarely share most interests. Although such a result is obtained from Facebook data, it reflects users' interests and matches our daily experience that people usually do not share many common interests [18].

O3: *In a social network, nodes usually have multiple interests. Two nodes sharing one interest usually do not necessarily share many other interests.*

Based on O1, O2, and O3, we can infer that:

I1: *Forming frequently contacted nodes into a community may not be able to limit most file searches within a community due to nodes' movement and diverse interests.*

I2: *Same interest nodes may not always connect to each other due to node mobility.*

3.3 Why Combined Content and Contact Design

Figure 3 shows an example of the communities constructed with a contact-based method (i.e., MOPS) and a content-based method (i.e., Cont^2), respectively. In MOPS, each community consists of nodes in a department since they meet frequently. In Cont^2, nodes are grouped according to their interests (contents). In the figure, some students (gray nodes) in different departments also attend a poetry class. These nodes gather together regularly during the poetry class.

Obviously, MOPS is efficient if most requests can be satisfied within its current community for its tight intra-community connections. However, this does not hold in practice (**I1**), leading to frequent inter-community searchers. Also, since communities in MOPS do not represent content information, a forwarder needs to know the content indexes of all other communities, which is costly and inefficient. For example, in Figure 3(a), a requester in community C_1 requests a file in community C_3. The broker of C_1 (b_1) has to know the file index of C_3 first. Otherwise, the file request either waits on b_1 for such information or gets forwarded blindly. Therefore,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems

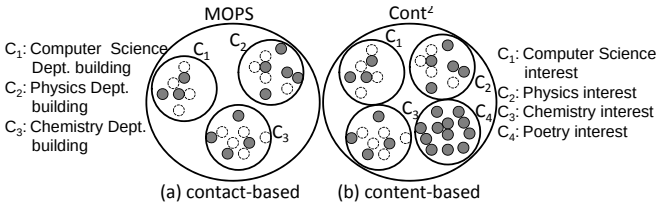


Fig. 3. Community creation of MOPS vs. Cont^2.

purely relying on node contacts for file search may lead to a low searching efficiency.

On the other side, nodes in interest (content) based communities may not tightly connected (**I2**). Therefore, a request reaching the destination community cannot meet the file holder easily. For example, in Figure 3(b), when a poetry related request arrives at community C_4, its file holder may stay in other places at the moment, leading to a long waiting time. In this case, social properties about node contacts (**P2**) should be utilized to forward the request to the file holder greedily, thereby overcoming the relatively loose structure in interest (content) based communities and enhancing file search efficiency.

In summary, both contents and contacts are necessary for efficient file searching in DTNs.

4 The Design of Cont^2

Following the findings in the previous section, we design three components in Cont^2: community creation, neighbor table construction and update, and content and contact based file search. We also propose advanced techniques that can further enhance file searching efficiency, thought at additional costs. In below subsections, we first introduce the network model for Cont^2 and then elaborate the technical components in detail.

4.1 Network Model and Application Scenarios

Cont^2 is proposed for DTNs in which mobile nodes are held by people and the social network properties indicated in the introduction section are presented. There are n mobile nodes in such a network, denoted by N_i ($i = 1, 2, \cdots, n$). N_i is also called node i in this paper. Each node holds some files, which can reflect the interests of the node. The interests represent the content categories of shared files in Cont^2. Therefore, the interests are specified by the file sharing applications running above Cont^2 and do not necessarily include all general file interests. Recall that based on the social network properties, nodes with common interests, i.e., interested in similar contents, tend to meet more frequently than with other nodes. However, Cont^2 does not require common-interest nodes to always stay together, and they can be dispersed in the network.

Considering the high packet routing delay in DTNs, Cont^2 is suitable for delay-tolerant file sharing rather than time-sensitive file sharing. It can be applied to many practical scenarios. For example, students on a campus can share video clips and study materials through Cont^2. People in rural villages, where it would be costly to build infrastructures, can share files based on the peer-to-peer communication between their devices in Cont^2.

4.2 Community Creation

Without loss of generality, we assume that files stored on a node can be reflected by the node's interests. We then define a *community* as a group of nodes sharing the same interest for the purpose of file searching. Thus, in Cont^2, common-interest nodes virtually form into a community. By virtually, we mean that nodes in one community do not have to always stay together. However, based on previous discussion, same interest nodes still are more likely to meet with each other than with others, which connects our community definition with node mobility for the purpose of file sharing. A node with multiple interests belongs to multiple communities.

Cont^2 has a server that functions as a bootstrap for node (or interest) joining. The server maintains an interest list containing all interests and associated keywords in the system. This list is initially configured by the system administrator and is updated when necessary. For example, in a movie file sharing system, the interests include action, comedy, and romance. The server can map a set of keywords to an interest. Such automatic mapping can be conducted through machine learning techniques, which are beyond the discussion of this paper. A node needs to register to the server when joining in the system. The node derives its keywords from its files through a keyword abstraction technique [25] and reports that to the server during the registration process. Based on the keywords, the server identifies the node's interests and community IDs. If there is no community matching the node's interest, a new community ID is created for the node. When a node changes its interests, it also reports to the server to get the new community IDs. A node can manually connect to the server through the Internet or 3G network for the registration, and then switches to the P2P mode for file sharing.

Common-interest communities benefit file searching from two aspects: (1) it increases the probability that a node finds its interested files in its own community since common-interest nodes tend to meet more frequently, as introduced in Section 1, and (2) it can enable a request to learn the destination community directly.

4.3 Neighbor Table Construction and Update

Each node in Cont^2 maintains a *neighbor table* (Table 2) to help decide the next hop node in file searching. We present how this table is constructed and updated on each node below. The notations used in this subsection are summarized in Table 1.

TABLE 1
Notations for Neighbor Table

Notation	Meaning
CCT_i	Community Contact Table of neighbor N_i
C_x	The x-th community
F_{inC_x}	Node N_i's n-hop contact frequency with community C_x

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems

4.3.1 Neighbor Table Construction

The *neighbor table* on a node records the information of its current neighbors and encountered same-community neighbors. The information includes node ID, community ID, content synopses, *community contact table* (CCT) and a connection bit (CB). The content synopses of a neighbor shows its contents, which are generated with the keywords of all its files. Each keyword k is associated with a weight w_k, which is the portion of files that contain the keyword on the node. Thus, a content synopses is represented by $<k_1, w_{k_1}; k_2, w_{k_2}; \cdots >$. The content synopses is updated when two nodes meet. CB is a Boolean value indicating whether the node is currently connected to the neighbor.

TABLE 2
Neighbor Table

Node ID	Community ID	Content synopses	CCT	CB
1	0x0001, 0x0010	$< k_1, w_{k_1}; k_2, w_{k_2}; \cdots >$	CCT_1	1
2	0x0008	$< k_1, w_{k_1}; k_2, w_{k_2}; \cdots >$	CCT_2	0
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

TABLE 3
Community Contact Table (CCT)

Community ID	1-hop contact frequency	2-hop contact frequency
0x0001	0.7	0.9
0x0010	0.2	0.7
0x0011	0	0.5
0x0100	0.6	0.2
$\cdots$	$\cdots$	$\cdots$

In Table 2, CCT_i denotes the CCT (Table 3) of neighbor N_i, which records its n-hop ($n = 1, 2, 3, \cdots$) contact frequency with each community. A node's n-hop contact frequency with a community represents its probability of connecting to the community through n hops. A node's probability of connecting to a community equals to its accumulated probabilities of meeting the members in the community. We use this definition since CCT serves to forward a file request to its matched community. Specifically, node N_i's 1-hop contact frequency with community C represents its direct contact probability with C; N_i's n-hop ($n > 1$) contact frequency with community C refers to the accumulated $n-1$ hop contact frequency of N_i's current and past neighbors with C. CCT helps inter-community search by selecting nodes with the highest probability of meeting the destination community as the next relay node. Although more information in the CCT would give better direction on request forwarding, we confine n to 2 in order to balance the routing performance and storage and transmission cost.

4.3.2 Neighbor Table Update

We use F_{inC_x} ($n \geq 1$) to denote node i's n-hop contact frequency with community C_x, which is initialized to 0 in the beginning. Node i periodically updates its F_{inC_x} with each community after each unit time period T. Specifically, when node i meets a new neighbor j, they exchange their neighbor tables for subsequent periodical table updates. Suppose the accumulated time period that nodes i and j connect with each other is t_{ij} during T and node j belongs to community C_1. Then, node i's CCT is updated by:

$$\begin{cases} F_{inC_1} = F_{inC_1} + \frac{t_{ij}}{T}; \text{ if } n = 1 \\ F_{inC_x} += \frac{t_{ij}}{T} * F_{j(n-1)C_x}; \text{ if } n > 1 \ \& \ x \neq 1 \end{cases} \quad (1)$$

where F_{inC_x} ($n > 1$) refers to node i's n-hop contact frequency with each community C_x. Thus, N_j's (n-1)-hop contact frequency with C_x is added to F_{inC_x} because a message from N_i needs one more hop to reach C_x through N_j. With such design, the calculated community meeting frequency may be larger than 1 since a node may meet multiple nodes from a community in each T. However, it still can reflect the relative tightness of a node's connection with a community.

In the end of each T, node i updates its contact frequency to each community by

$$F^{new}_{inC_x} = \beta F^{old}_{inC_x} + (1-\beta) F_{inC_x} \ (n \geq 1), \quad (2)$$

where $\beta < 1$ is a fading weight, $F^{old}_{inC_x}$ and $F^{new}_{inC_x}$ denote node i's old and new contact frequency with community C_x before and after T, respectively. The value of β is determined by the weight of previous and most recent meeting frequency on deciding F_{inC_x}. The system administrator should decide a suitable T based on the mobility of nodes in the system.

The *community contact table* reflects a node's contact frequencies with different communities and guides the file searching algorithm, as explained later. Therefore, it should be updated properly so that it can reflect both long term meeting ability and short term changes. β is designed for this purpose by controlling how fast the overall contact frequency evolve along with the contact frequency measured in current time unit. It should match the actual frequency change rate. We leave how to decide it accurately to future research. In this paper, considering meeting frequencies in daily social network usually present both long term stability and short term changes, we set β to a medium value of 0.5.

4.3.3 Functions of the Neighbor Tables

With the above design, a node's neighbor table provides information on how it can meet each community and the synopses of contents hold by nodes in its community. The former helps guide a content request to reach the community that contains the requested file. The latter helps determine which nodes contain the requested file, i.e., possible file holder. Then, the request is routed to reach the possible file holder based on above information. Even if the possible file holder does not have the requested file, it can provide more information on which node may hold the requested file since nodes containing similar contents tend to meet with each other. In summary, the neighbor table provides organized information regarding how to find the requested file efficiently.

4.3.4 Efficient Storage of the Neighbor Tables

Considering that the storage resource on mobile nodes usually is limited, we further propose a strategy to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems

improve the storage efficiency of the neighbor tables. When a node, say N_i, is disconnected with its neighbor node, say N_j, N_i cannot forward a request to N_j anymore. Then, N_i does not need to store the CCT of N_j, which is mainly used to guide the request forwarding. Thus, we let the neighbor table only store the CCTs of currently connected neighbor nodes and the CCTs of frequently met same-community nodes. Always storing the CCTs of frequently met same-community nodes avoids frequently adding and removing these CCTs, which otherwise would cost significant communication cost. Similarly, each node also stores the content synopses of a number of top frequently met nodes from other communities in order to save the communication cost. Then, when node N_i meets its top frequently met node, say N_k, N_i can use its stored content synopsis of N_k in file searching. If the content synopsis has been updated after last encountering, N_k informs N_i.

4.4 Content and Contact Based File Searching

The searching algorithm is developed based on the social network property described in Section 1. Since each node knows the interest list in the system, a file requester can map its request to an interest (destination community). When a node receives a request, if it is the file holder, it returns the file. Otherwise, if the node is located in the destination community, it conducts intra-community searching. If it is not in the destination community, it conducts inter-community searching, which forwards the request gradually to the destination community. When the request arrives at the destination community, the intra-community search is launched. Note that the definition of community ensures that the requested file, if exists in the system, is highly possible to be held by nodes in the destination community (i.e., nodes with files in the interest are classified into the corresponding community). Then, it is not needed to request encountered nodes for the requested file before arriving at the destination community, thereby saving energy for file searching. Each request has a Time To Live (TTL), after which the request is expired and is dropped.

4.4.1 Intra-Community Searching

In this step, requests are forwarded within the destination community to find the file holder. In each forwarding, the request is forwarded to a neighbor node that has more intra-community connections toward the node having the highest similarity with the requests file. Such a design comes from two reasons. First, the node having a high similarity with a request has high probability of containing the requested file. Second, an interest can usually be further classified into sub-interests, and people in a sub-interest group have a higher probability of meeting with each other than with other members in the interest community. For example, lab members majoring in computer systems tend to meet more often. Then, even when the high similarity node fails to satisfy the request, its frequently met nodes may contain the requested file. Therefore, the similarity works as an indication of the probability of satisfying the request.

We denote the node with the highest similarity with the requests file as the *temporary destination node (TDN)*. The similarity is calculated as following:

$$Sim(R_f, N_i) = \sum_{k \in K} w_k, \tag{3}$$

where R_f is the request, K denotes the keyword group in the request, and w_k denotes the weight of keyword k in node N_i's content synopses. w_k=0 if the synopses does not contain k. The similarity here shows the percentage of files on the node that matches the keywords in the request and indicates the possibility that the requested file is on the node. Therefore, the TDN should be the node with $\max\{Sim(R_f, N_i)\}$ among all nodes that have been visited by the request. That is

$$Sim(R_f, TDN) = \max\{Sim(R_f, N_i)\ (N_i \in AN(R_f)\}, \tag{4}$$

where $AN(R_f)$ represents all nodes that R_f has already visited. When a request arrives at a new node, the $AN(R_f)$ and TDN are updated accordingly.

Specifically, suppose node N_a receives the file request, if it holds the requested file, it returns the file to the requester and the file search is successful. Otherwise, it first checks whether the file holder or the TDN currently is connected. If yes, the file holder or the TDN node is the next relay node. If not, by referring to the CCT in its neighbor table, N_a then chooses the node (i.e., active node) that is lightly loaded and has the highest F_{i1C} with its current community as the next relay node. This is because the node with the highest F_{i1C} has more contacts with the destination community and usually has higher probability to meet the TDN or know the actual file holder. We consider node load status in routing because the active node may become overloaded. A node measures its overload status by the occupation of its buffer, and piggybacks such a status on the "hello" messages used in neighbor scanning process.

4.4.2 Inter-Community Searching

In inter-community searching, node N_a first checks its neighbor table to see whether there is a neighbor from the destination community (C_d), and takes it as the next relay node if one exists. If more than one exist, N_a chooses the one with the highest F_{i1C_d} by referring to the CCT in its neighbor table since that node has more connections with the destination community. If none of such nodes can be found, N_a chooses the node that has the highest F_{i1C_d} as the next relay node. If multiple nodes have the same highest F_{i1C_d}, the node with the highest F_{i2C_d} is chosen. In this way, the request can be quickly forwarded to the destination community, because F_{inC} reflects a node's probability of meeting nodes in community C in n hops.

It is possible that all nodes in current community have very few contacts with the destination community. To deal with this problem, we pre-define a threshold for F_{i1C} and F_{i2C}, denoted by T_d. If no node in the neighbor

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems

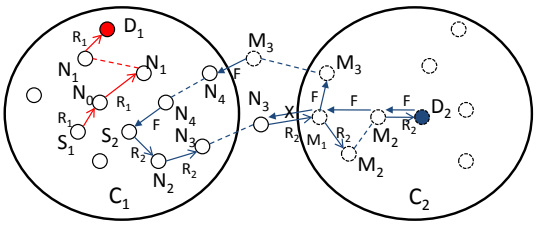


Fig. 4. File searching and retrieval process.

table has $F_{inC} > T_d$, the active node with the highest 1-hop contact frequency with the current community is chosen. The purpose of this strategy is to quickly move the request out of current area. If the node itself is chosen as the next relay node after these steps, it holds the request. While it is moving, it updates its neighbors and repeats the above steps until the request is forwarded to the destination community.

4.4.3 File Retrieval and Summary

Upon receiving a request, the destination node first tries to send the file back along the route the request traversed, which is inserted into the request during the forwarding process. If the reverse route is broken, the intra-community and inter-community searching algorithms can be used to send the file back to the requester according to the IDs of the requester and its community. In current design, we only consider the scenario that there is only one matched file for each request. This can be extended to multiple matched requests by allowing each request to continue searching after a successful hit.

Figure 4 shows an example of file searching between two communities (C_1 and C_2). Two requests (R_1 and R_2) are initiated in C_1. R_1's destination community is C_1. Thus, the requester uses intra-community searching. It first finds the temporary target N_0 in its neighbor table and forwards R_1 to N_0. N_0 updates the temporary target with D_1, which has closer similarity with the requested file. Because D_1 is not N_0's current neighbor, N_0 forwards R_1 to its neighbor N_1, which has the highest 1-hop contact frequency with C_1. Since D_1 is not N_1's current neighbor, N_1 holds the request, and delivers R_1 to D_1 when moving close to it. D_1 notices that its synopses match the request exactly, so it replies the requested file to the requester.

The file requested by R_2 belongs to community C_2. Since the requester cannot find a current neighbor that belongs to C_2, it forwards R_2 to its current neighbor N_2 that has the highest 1-hop contact frequency with C_2. N_2 forwards R_2 to N_3. On the way moving to C_2, N_3 forwards R_2 to M_1 of C_2, which has higher 1-hop contact frequency with C_2. Then, intra-community searching is used to forwards R_2 to destination D_2 through M_2.

We use a line with a solid arrow to stand for the file retrieval process. Node D_2 first sends the requested file (F) back to the requester S_2 along the request route. However, when M_1 receives the request, it finds that node N_3 on the reverse route is not available. Node M_1 then launches a search for requester S_2. First, inter-community searching is used to route F to node N_4 in community C_1 through node M_3 in community C_2. Second, N_4 starts intra-community searching. N_4 finds itself has the highest mobility among all neighbors, so it carries the request until it moves close to S_2.

4.5 Advanced File Searching

The three components introduced above can realize efficient file searching. However, they suffer from two drawbacks considering sparse node distribution in DTNs. First, interest-based community structure may have a loose structure, which limits the efficiency of the intra-community search. Second, a request may be generated in an area that is far away from the file holder, which means that it may not be forwarded towards the correct direction in the beginning. We then propose advanced techniques to solve the two problems and further improve the efficiency of file searching in Cont^2. Specifically, we exploit contact-based sub-communities within each interest community to facilitate intra-community file searching. We also propose a parallel forwarding algorithm to improve the inter-community file searching efficiency. We present the details below.

4.5.1 Contact-based Sub-communities

Recall that common-interest nodes form a community. Actually, nodes within the same community also form sub-communities, in which nodes meet with each other more frequently than with other community members. This phenomenon is normal in real life. For example, as shown in Figure 5, for graduate students with the Computer Science interest, students in different labs meet their labmates more frequently than with those in other labs. Clustering nodes that more frequently meet each other can expedite forwarding requests to file holders. Therefore, we further propose an algorithm to construct contact-based sub-communities in Cont^2 to facilitate file searching.

Specifically, each node first collects its meeting frequencies with same-community members. After nodes have collected their stable contact frequencies with same-community members, i.e., after the initial period, they begin to construct sub-communities. Each sub-community has a coordinator known by all sub-community members that is responsible for membership management. It is the node in the sub-community that can meet the most nodes in the sub-community.

We define a node's *average contact frequency* with a group of nodes as the sum of its contact frequencies with these nodes divided by node number. It is used for sub-community construction. In detail, when two nodes, say N_i and N_j, meet with each other, if they both do not belong to any sub-communities, they check whether their contact frequency is more than T_{sub} times of their individual average contact frequency with other community members. If yes, they form a new sub-community and the node with higher contact frequency with the whole community (i.e., F_{i1c}) is selected as the coordinator. If one of the two nodes, say N_j, is the coordinator of one

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems

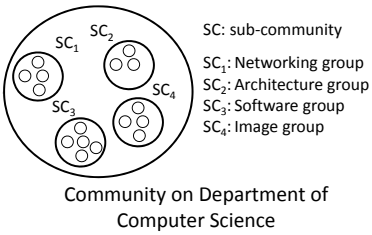


Fig. 5. Demonstration of sub-communities.

sub-community, and the other node N_i does not belong to any sub-communities, N_j checks whether N_i can be a member of its sub-community. If N_i's average contact frequency with N_j's sub-community is more than T_{sub} times of its average contact frequency with nodes in the community excluding N_j's sub-community, N_i becomes a member of this sub-community. If N_i's average contact frequency with the community is larger than N_j's, N_i becomes the new coordinator for this sub-community and N_j notifies the sub-community members about the coordinator change.

When a node notices that its average contact frequency with another sub-community (denoted by SC_j) is higher than its current sub-community (denoted by SC_i), it needs to leave SC_i and joins SC_j. To avoid frequent sub-community joins and departures, we set a threshold T_{sj}, (e.g., 20%). Only when its average contact frequency with SC_j is T_{sj} more than that with SC_i, the node leaves SC_i and joins SC_j. The node then notifies SC_i's coordinator to remove its membership. By this process, contact-based sub-communities are gradually created in each interest community. The values of T_{sub} and T_{sj} are determined by the dynamism degree of contact frequencies in the community. If the contact frequencies among nodes are stable, small T_{sub} and T_{sj} can help find sub-community effectively. If the contact frequencies change frequently, larger T_{sub} and T_{sj} are needed to avoid false positives on sub-community construction.

4.5.2 Advanced Intra-community Searching

The contact-based sub-communities help enhance the efficiency of intra-community file searching. In the intra-community searching algorithm introduced in Section 4.4.1, the node with higher F_{i1C_d} (i.e., with more connections with the destination community) is selected as the carrier for the request. However, such a node may not have a high probability of meeting the file holder. It may frequently meet nodes in its sub-community but rarely meets nodes out of its sub-community. Consequently, a request may be trapped in the sub-community, which degrades the file searching efficiency.

In order to solve this problem, we propose another metric for intra-community file searching: *active level*. It represents the frequency that a node visits different sub-communities in a community. We use $F(SC_{ik}^j)$ to denote N_i's contact frequency with the j-th sub-community in C_k, which is calculated as the average portion of a unit time period that N_i connects with the j-th sub-community. We use such a definition to limit the contribution of the connection with one sub-community in the overall active level. We use AL_{ik} to denote N_i's active level in community C_k, which is calculated as

$$AL_{ik} = \sum_{j=1}^{v_{ik}} F(SC_{ik}^j) \tag{5}$$

where v_{ik} is the total number of sub-communities N_i has contacted. Therefore, a larger AL_{ik} means that N_i visits more sub-communities in C_k on average.

The active level is used to decide the carrier for a file request in the advanced intra-community searching algorithm. Specifically, after a file request arrives at the destination community, it is first forwarded to the node with higher active level in the destination community. This can enable the file request to meet more sub-communities. When the request arrives at a node in a sub-community, it can learn whether the file holder exists in the sub-community from the content synopses in the neighbor table of the node. Then, the request can quickly find the sub-community that contains the holder of the requested file through the high active level node. Once the sub-community is identified, the file request is only forwarded to the node that has higher average contact frequency with the sub-community.

Such a scheme can improve the intra-community searching efficiency since the node with high active level can carry the file request to more sub-communities and consequently enhances its probability of meeting the file holder or knowing which node is the file holder.

4.5.3 Advanced Inter-community Searching

Recall that in the inter-community searching process introduced in Section 4.4.2, when none of the nodes in the neighbor table satisfies $F_{inC} > T_d$, a file request is forwarded to the node with the highest 1-hop contact frequency with the current community that the request resides in. However, this scheme cannot ensure that the file request can arrive at the destination community quickly. Therefore, we propose parallel forwarding to enhance the efficiency of this step.

Specifically, when none of the nodes in the neighbor table satisfies $F_{inC} > T_d$, the request is allowed to be replicated to a number of nodes to enhance its probability of being forwarded to the destination community. Two types of nodes can be the recipients of the replicas of a request. The first one has a high active level in the community, which can help carry the request to other sub-communities to search for better forwarder. The second one has a high overall contact frequency with other communities. Such a node can carry the request to other communities to search for a better forwarder. A request replica is not further replicated to constrain the overhead. When a file holder receives multiple replicas of the same request, it only responds to the first one and ignores all others. Note that this does not mean that the holder only sends one response for the request. When necessary, it can send multiple responses for a request to ensure that the requester can receive the file.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems

The replications of a file request can help forward it to the destination community quickly, thereby avoiding being trapped in a community. Consequently, the overall file searching efficiency is improved.

5 Performance Evaluation

We first deployed the systems on the GENI Orbit testbed [26], [27]. The testbed has 400 nodes in total, each of which is equipped with a wireless card. We adopt such a testbed since it can reflect the performance of Cont^2 in a more real scenario. We then conducted experiments on NS-2 [28] using the converted one-day trace data since the whole trace is too long for simulation. We also used a community based mobility model [29] to further evaluate the systems with different network sizes and node mobility. Detailed introduction and configuration of the community based model are given later in Section 5.1.4. We also evaluated the performance of the advanced file searching algorithm, which is introduced in Section 4.5, in Section 5.6. Considering the storage and energy constraint of mobile devices, we expect the system size to be at most several hundreds of nodes.

5.1 Experiment Settings

In order to better evaluate the file search, we determined the community construction and file generation in advance. We extracted interest groups and corresponding keywords from the profiles we crawled from 117 Facebook users who watched the same randomly selected video. We selected the top seven mostly shared interests and mapped them to the 7 communities identified from the real trace. For each of a node's interests, we randomly selected 40 keywords from the keyword pool of the interest and generated about 20 files. A requested file of an interest is randomly picked from the file pool of the interest, and each request matches only one file in the system. Consider people would like to request file in its interests, 70% of requested files have the same interest of the originator. We run each test for 5 times and present the average results within the 95% confidence interval. The confidence intervals are very small and thus are not shown in figures to avoid obscuring the results.

5.1.1 Comparison Methods

We used below comparison methods:

(1) MOPS [8]: MOPS provide publish/subscribe service in disruption-tolerant networks by grouping nodes that have frequent contacts into a community, which is regarded as a reflection of social relations. Each community uses nodes that have frequent contact with other communities as brokers for inter-community communication. We use 2 brokers for each community to balance the cost and search efficiency.

(2) PDI+DIS [30], [31]: PDI [31] uses 3-hop local broadcast and content tables for file searching. A content table contains corresponding routes to content owners and is built in nodes along the response path. We complement PDI with the advertisement-based DIS method [30], in which each node disseminates its content to its neighbors. In PDI+DIS, a request is first broadcasted for 3 hops. During and after the 3-hop broadcasting, when it finds a route to the requests content, it follows the route for file searching. A node buffers a request if it cannot forward the request due to expired routes or lack of routes.

(3) Epidemic [32]: Epidemic is a buffering based broadcasting mechanism. It tries to disseminate requests to all nodes in the system by letting two nodes exchange requests they haven't seen upon their encountering.

5.1.2 Test Metrics

Since the routing process of file retrieval is similar as the file searching process, we only test the performance of file searching. We used the number of generated messages to represent costs. We define that one message only contains the information of one node.

(1) *Hit rate:* the percentage of requests that are successfully delivered to the file holders.

(2) *Average delay:* the average delay of all requests that successfully find the requested files.

(3) *Maintenance cost:* the total number of message forwarding for routing information update (i.e., node content exchange in all the four methods, request exchange in Epidemic, routing table establishment in PDI+DIS, and neighbor table construction in Cont^2).

(4) *Total cost:* the total number of messages generated during the simulation including request messages.

5.1.3 GENI Experiment Parameters

We first conducted experiments on the GENI Orbit testbed, which consists of 400 nodes that can connect with each other through wireless connections. We adopt the MIT Reality project trace to drive node mobility, which lasts for about 2.56 million seconds. We set the first 0.3 million seconds as the initialization period so that nodes can collect enough records to reflect their meeting probabilities with others. After that, we randomly picked one node to generate one request every 100 seconds for 1 million seconds. The watching period (or TTL) of each request was set to 1.2 million seconds. To be practical, each node can hold at most 2000 requests in its buffer. When the buffer is full, the oldest request is dropped.

5.1.4 Simulation Parameters

According to the settings in the GENI experiment and the works in [33]–[35], we determined the parameters for the simulation on NS-2, as shown in Table 4.

We recorded the experimental metrics every 100s after the initialization period. In the community based mobility model [29], the test area is divided into many caves, each of which represents one community area. Nodes move within its home community randomly in most of the time. We map each interest community defined in Cont^2 to a randomly selected cave in the mobility model [29]. The model also allows setting travelers

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems

TABLE 4
Parameters in simulation

	Real trace	Synthesized
Environment Parameters	**Value**	**Default Value**
Simulation area	$2.5km \times 2.5km$	$2.5km \times 2.5km$
Number of caves	-	25
Cave size	-	$500m \times 500m$
Number of communities	7	7
Node Parameters	**Value**	**Default Value**
Number of nodes	45	100
Communication range	250m	250m
Average node speed	–	$1.5m/s$
Re-wiring probability	–	0.1
Number of travelers/cave	–	2
Traveler speed	–	2*(average speed)
Number of keywords	40	40
Requesting Parameters	**Value**	**Default Value**
Requesting rate	$8/s$	$8/s$
Intra-request percentage	70%	70%
Initialization period	1000s	1000s
Requesting period	2000s	2000s
Waiting period	51000s	51000s

that frequently commute between two communities with high speed. We then take travelers as brokers in the test. Considering that travelers are more active, we set a normal node's speed to a value randomly selected from the range $[2v/3, 4v/3]$ (v denotes the average speed) and a traveler's speed to $2v$. In the tests with different node mobility, we varied the average speed (v) from the medium walking speed of human beings ($1.5m/s$) to the medium vehicle speed ($15m/s$).

5.2 Performance in GENI Experiment

Table 5 shows the GENI experiment results. We see that Epidemic generates the highest hit rate but also the highest cost since it tries to replicate each request to all nodes. PDI+DIS generates the lowest hit rate because the routes in content tables often expire due to node mobility and many requests wait passively for the file holders. Therefore most successful requests in PDI+DIS are resolved locally within 3 hops, leading to a low average delay. Cont^2 achieves the second highest hit rate but significantly lower cost than Epidemic. Cont^2 also generates higher hit rate than PDI+DIS, which shows Cont^2's higher mobility-resilience. Comparing Cont^2 and MOPS, we see that Cont^2 is superior over MOPS in terms of hit rate, delay and cost. This is because Cont^2 utilizes all possible forwarding opportunities around a node while MOPS relies heavily on brokers.

We also evaluated the memory usage of the four methods, measured by the average number of requests in the buffer and the average size (i.e., number of entries) of the neighbor table. Table 6 shows the average values of all nodes. We find that PDI+DIS has the smallest average number of requests in the buffer. This is because most requests are quickly resolved in the 3-hop local broadcasting without further buffering. In Cont^2 and MOPS, requests are buffered until meeting better forwarding nodes. Cont^2 generates fewer requests in the buffer than MOPS since it can deliver requests more quickly as shown in Table 5. Epidemic produces the largest memory usage as it tries to distribute each request to all nodes.

Each node in all methods except the Epidemic also stores the content synopses of nodes it has met. Cont^2 and MOPS need content synopses for both intra- and inter-community searches. PDI+DIS uses it to build content tables. From Table 6, we find that Cont^2 stores fewer content synopses than the other two methods because it only stores the information of same community nodes and currently connected neighbors. For MOPS, brokers store that of all communities they have known, and normal nodes store that of the nodes in their own communities. Therefore, MOPS has the largest average number of stored content synopses. In PDI+DIS, a node disseminates its contents to its neighbors. Thus, each node stores the content synopses of all nodes it has met, leading to higher memory consumption than Cont^2.

In summary, Table 5 and Table 6 show that Cont^2 has superior performance over other methods considering overall efficiency, delay, cost and memory consumption.

TABLE 5
Efficiency and cost in the experiments on GENI

Method	Hit Rate	Average Delay (s)	Maintenance Cost	Total Cost
Cont^2	0.696	142892.8	231918	269917
MOPS	0.625	161070.0	311302	328266
PDI+DIS	0.508	7562.5	301918	361506
Epidemic	0.8745	15230.1	676685	867939

TABLE 6
Memory usage in the experiments on GENI

Metric	Cont^2	MOPS	PDI+DIS	Epidemic
Ave. # of requests in buffer	33.5	43.5	12.3	1998.6
Ave. size of a neighbor table	7.9	17.5	15.7	0

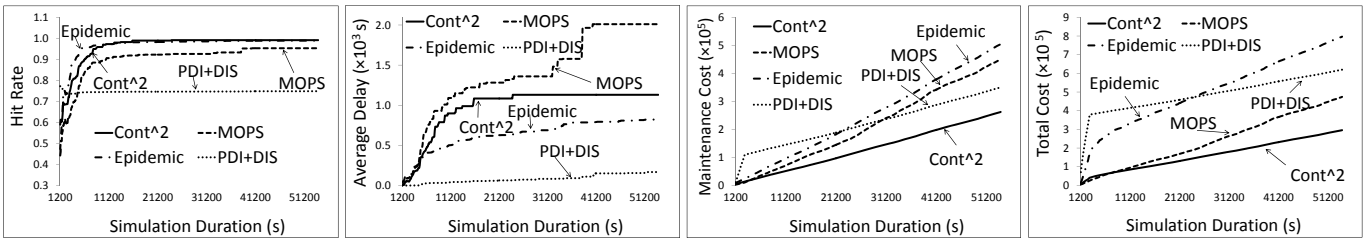
5.3 Performance in Trace-drive Simulations

5.3.1 Hit Rate

Figure 6(a) shows the hit rates of different methods over time. We see that the hit rates of Epidemic and Cont^2 reach 99%, that of MOPS is about 95% and that of PDI+DIS is below 90%. Epidemic tries to disseminate each request to all nodes in the system, thereby resulting in a high hit rate. In Cont^2, request forwarders fully utilize nearby nodes to forward the request in the direction of the destination, leading to efficient file search.

MOPS only relies on brokers for inter-community search and direct encountering of community members for intra-community search. As previously introduced, due to the diversity of node interests, inter-community search is always needed in MOPS. However, without the guidance of content, MOPS has to rely on the information exchange between brokers for inter-community search, which may miss some forwarding opportunities. Moreover, the passive waiting in the intra-community search also takes a long time. Consequently, MOPS cannot resolve some requests before they expire, resulting in a lower hit rate than Cont^2. PDI+DIS only completes about 74% of requests, and its hit rate remains nearly constant throughout the test. This is because many requests cannot be resolved in the test since the routes in a content table expire quickly due to node mobility. After the local broadcast, requests just wait passively, leading to almost no increase in the hit rate.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems



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

Fig. 6. Performance in trace-driven simulation.

We also find that the hit rates of Epidemic, Cont^2, and MOPS exhibit sharp rises at the initial stage and increase slightly afterwards, while PDI+DIS remains nearly constant throughout the test. In Epidemic, Cont^2, and MOPS, requests that cannot be immediately resolved stay in current nodes and gradually arrive at file holders using request forwarding algorithms. Therefore, the hit rate increases gradually. In PDI+DIS, after 3-hop broadcasting, buffered requests passively wait for routes to file holders, generating much fewer successful searches. Therefore, its hit rate remains almost stable.

5.3.2 Average Delay

Figure 6(b) shows that the average delays of the four methods follow MOPS>Cont^2 >Epidemic>PDI+DIS. Recall that we only measure the delay of successful requests. So PDI+DIS has the least average delay since most successful requests are resolved in the initial 3-hop broadcasting stage. Cont^2 reduces the delay of MOPS by half for the same reasons as in Figure 6(a). This result confirms that only considering contacts for file search cannot provide high efficiency. Epidemic results in relatively lower average delay than Cont^2 due to its broadcasting nature. It is reasonable that Cont^2 generates higher delay than Epidemic since Cont^2 only maintains one copy for each request.

It is interesting to observe that the delay of MOPS increases rapidly at around 40000s while that of Cont^2 increases steadily. In MOPS, a broker may need to buffer inter-community requests for a long period before being able to contact brokers in a neighboring community. Then, the encountering of two brokers with many unresolved requests may lead to a rapid increase in hit rate and average delay, as occurred at 40000s. Without relying on fixed brokers, Cont^2 utilizes every forwarding opportunity, leading to a steady increase in hit rate and average delay. This result confirms the drawback of depending only on brokers and the advantage of utilizing all available nodes in request forwarding.

5.3.3 Maintenance Cost

Figure 6(c) plots the maintenance costs of different methods over time. After 31200s, the costs follow Epidemic>MOPS>PDI+DIS>Cont^2. In all methods, node content is exchanged among encountered nodes, which contributes to the linear growth of maintenance cost over time. Except Cont^2, nodes in other three methods need to exchange other information in addition to their own contents. PDI+DIS builds routes in nodes along the response paths of successful requests. Thus, PDI+DIS generates higher maintenance cost than Cont^2. In MOPS, brokers exchange contents of all nodes from their home communities, leading to an even higher maintenance cost. In Epidemic, two nodes exchange information about all known requests to decide unseen requests, resulting in the highest maintenance cost.

5.3.4 Total Cost

Figure 6(d) shows the total cost of each method over time. We observe that Epidemic>PDI+DIS>MOPS >Cont^2. Epidemic has very high cost since it tries to replica a request to all nodes. Each message has only one copy in MOPS and Cont^2. PDI+DIS has a local broadcast, which generates many request copies, leading to higher total cost than MOPS and Cont^2.

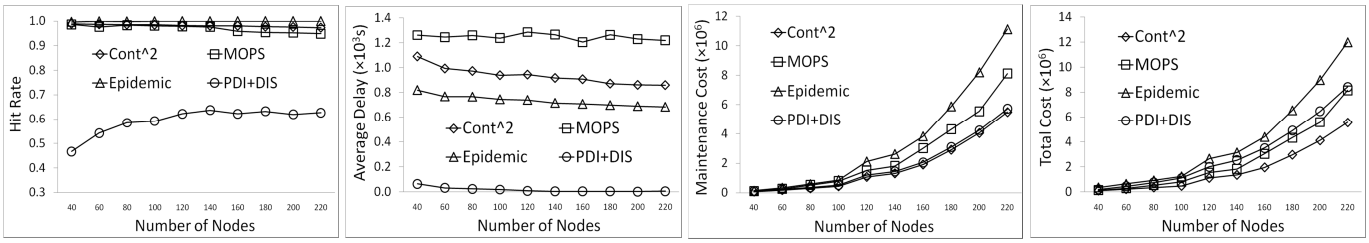
5.4 Performance With Different Network Sizes

In this test, we evaluate the four methods when the total number of nodes varied from 40 to 220 in Simulation. When the number of nodes increases, the total amount of data in the network also increases.

5.4.1 Hit Rate

Figure 7(a) plots the hit rates of the four methods. We find that the hit rates of Epidemic, Cont^2 and MOPS reach over 95% (MOPS<Cont^2<Epidemic) while PDI+DIS resolves only 60% of requests. Epidemic achieves a high hit rate due to its system-wide message dissemination. The reasons for MOPS<Cont^2 and the low hit rate of PDI+DIS are the same as explained in Figure 6(a). Also, the hit rates of Cont^2, MOPS, and Epidemic remain relatively stable while that of PDI+DIS increases as the network size increases. The former three methods actively forward messages to file holders by broadcasting or routing algorithms, which ultimately forwards most requests to their destinations, even in a sparse network. In contrast, unsolved requests after the 3-hop broadcasting stage in PDI+DIS passively wait for routes to the file holders. Hence, higher node density enables more forwarding opportunities, thus increasing the hit rate of PDI+DIS.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems



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

Fig. 7. Performance with different network sizes in simulation.

5.4.2 Average Delay

Figure 7(b) shows the average delay of each method. The result matches what was obtained in Figure 6(b), i.e., MOPS>Cont^2>Epidemic>PDI+DIS and Cont^2 reduces the delay of MOPS by about 20% on average. We also find the delay of MOPS remains relatively stable while those of Epidemic and Cont^2 exhibit a slight decrease as network size increases. This is because higher node density means more forwarding opportunities, thereby reducing the delay. MOPS only relies on brokers for inter-community communication. Though the number of nodes increases, the probability that two brokers meet does not change. Therefore, the average delay of MOPS remains stable though the network size increases.

5.4.3 Maintenance Cost

Figure 7(c) illustrates the maintenance cost of each method. It can be observed that the maintenance costs follow Epidemic>MOPS>PDI+DIS>Cont^2. Generally, Epidemic produces much higher maintenance cost than Cont^2, the maintenance cost of MOPS is approximately 40% higher than that of Cont^2, and PDI+DIS generates a 5% higher maintenance cost than Cont^2 in dense networks. The reasons for this result remain the same as in Figure 6(c). Also, the costs of all the methods grow quickly as network size increases. This is because with more nodes in the network, the amount of exchanged messages increases.

5.4.4 Total Cost

Figure 7(d) demonstrates that Epidemic generates the highest total cost. PDI+DIS and MOPS show moderate total costs while Cont^2 has the lowest total cost. Epidemic has the highest total cost since it generates a high maintenance cost (Figure 7(c)) and a large amount of requests in the test. The number of request messages in PDI+DIS increases quickly as the number of nodes increases due to the local broadcasting, resulting in a high total cost. MOPS incurs approximately the same number of request messages as Cont^2, but renders higher maintenance cost (Figure 7(c)), leading to higher total cost than Cont^2. The results in Figure 7 show the superior performance of Cont^2 and its efficiency in networks with different sizes.

5.5 Performance With Different Node Mobility

We evaluated the performance of the methods when the average speed varied from walking speed ($1.5m/s$) to medium vehicle speed ($15m/s$) in Simulation.

5.5.1 Hit Rate

Figure 8(a) shows the hit rate of each method. Epidemic, Cont^2, and MOPS have hit rates close to 100% at all speeds while PDI+DIS exhibits a low hit rate. The results show the same relative performance of the four methods as in Figure 6(a) and Figure 7(a) with the same reasons.

5.5.2 Average Delay

In Figure 8(b), the average delays of the methods follow MOPS>Cont^2>Epidemic with PDI+DIS having almost no delay at all node movement speeds. This matches with the results in Figure 6(b) and Figure 7(b) due to the same reasons. Moreover, we find that Cont^2 has a 20% lower delay than MOPS at all movement speeds, which confirms the high efficiency of Cont^2 in file searching. Also, we see that the delays of MOPS, Cont^2 and Epidemic decrease as node movement speed increases. This is because fast node movement increases the frequency of node encountering and reduces the waiting time of requests in the buffers, hence shortening the average delay.

5.5.3 Maintenance Cost

Figure 8(c) shows that the maintenance costs of the four methods follow Epidemic>MOPS >PDI+DIS>Cont^2 for the same reasons as in Figure 6(c) and Figure 7(c). We also find that the maintenance costs of Epidemic, PDI+DIS, MOPS and Cont^2 increase linearly when nodes move faster. This is because with faster movement, nodes meet frequently and exchange more contents, leading to a higher maintenance cost.

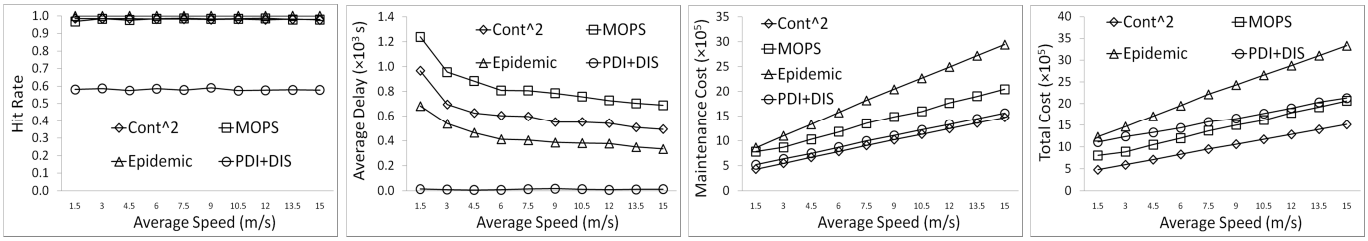
5.5.4 Total Cost

Figure 8(d) shows that the total costs of the four methods follow Epidemic>PDI+DIS >MOPS>Cont^2. The reasons are the same as in Figure 6(d) and Figure 7(d). The total costs of the four methods increase as the average speed increases because nodes generate higher maintenance costs with faster movement (Figure 8(c)). The total cost of Cont^2 still remains lower than other methods, which verifies Cont^2's low cost in different mobility rates.

5.6 Evaluation of the Advanced File Searching

In this section, we evaluate the proposed advanced file searching algorithm (Section 4.5). We set T_{sj} to 20% and the number of allowed replicas to 2. These values

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems



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

Fig. 8. Performance with different node mobility in simulation.

are chosen randomly in their reasonable ranges. The selection of parameter values will not change the relative performance differences between different methods.

5.6.1 Sub-community Construction

We first evaluate the contact-based sub-community construction algorithm (Section 4.5.1). We applied this algorithm to the 7 communities identified from the MIT Reality trace. We tested the algorithm with T_{sub} = 1.5, 2 and 2.5, respectively. Figure 9(a) shows the number of sub-communities identified in each community. We find that when T_{sub} increases, the number of identified sub-communities decreases. This is because when the threshold T_{sub} for joining a community increases, the contact frequencies between most nodes may not be qualified to form a sub-community. We also see from the figure that even when $T_{sub} = 2.5$, sub-communities can be identified in almost all interest communities. The results demonstrate the existence of sub-communities in real scenarios, which can be leveraged for more efficient file searching, as shown in the next subsection.

5.6.2 Advanced Intra- and Inter- File Searching

We conducted event-driven experiments with the MIT Reality trace. We followed the same configuration as in the GENI experiment but varied the number of file requests from 5000 to 25000. We used Cont^2 to represent the original Cont^2, Cont^2-Sub to represent Cont^2 with the advanced intra-community searching algorithm, and Cont^2-Adv to represent Cont^2 with both advanced intra- and inter-community file searching algorithms.

Figure 9(b) shows the hit rates of the three methods. We find that the hit rates follow Cont^2<Cont^2-Sub< Cont^2-Adv. This means that both the advanced intra- and inter-community searching algorithms improve the efficiency of file searching. We also find that Cont^2-Adv only slightly increases the hit rate compared to Cont^2-Sub. This is because only the requests that cannot find a suitable forwarder are replicated in Cont^2-Adv, which only count for a small portion of requests.

Figure 9(c) shows the average delays of the three methods. We find that the Cont^2-Sub and Cont^2-Adv present slightly higher average delay than Cont^2. This is because the advance file searching algorithms lead to more successful requests with a large delay, which may not be delivered successfully in the original Cont^2. Figure 9(d) shows the the average delay of all requests by counting the delay of a failed request as the configured TTL. Note that the TTL is smaller than the experiment time. Since Cont^2-Sub and Cont^2-Adv lead to more successful file requests, the overall delay is decreased. Above experimental results demonstrate the efficiency of the proposed advanced file searching algorithms.

6 Conclusion

This paper presents a content and contact based file search method for DTNs in a social network environment, namely Cont^2. It exploits the properties of social networks to enhance file searching efficiency. Through the study of a real trace, we found that the interests (content) of each node can help guide file searching. We also find that the movement patterns of mobile nodes can more accurately predict the encountering of nodes holding the requested files. Thus, Cont^2 virtually builds common-interest nodes into a community and forwards a file request to nodes with higher meeting frequency with the interest community or the node that has the most similar content with the requested file. We compared Cont^2 with other file search methods using mobility from both real-trace and a community based mobility model on the real-world GENI testbed and NS-2 simulator. Cont^2 shows superior performance in hit rate, delay and overall cost. In the future, we plan to investigate how the influence of a node's interest weights on its movement patterns and how to leverage it to enhance file search efficiency.

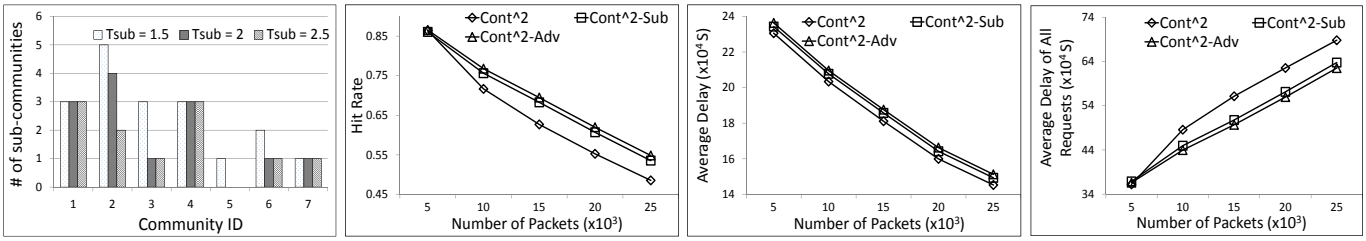
Acknowledgements

This research was supported in part by U.S. 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 ICPP'13 [36].

References

[1] "Haggle Project," http://www.haggleproject.org/t.
[2] M. Li, Z. Yang, and W. Lou, "Codeon: Cooperative popular content distribution for vehicular networks using symbol level network coding." *IEEE JSAC*, vol. 29, no. 1, pp. 223–235, 2011.
[3] K. Chen and H. Shen, "Dtn-flow: Inter-landmark data flow for high-throughput routing in dtns." in *Proc. of IPDPS*, 2013.
[4] S. Jain, K. R. Fall, and R. K. Patra, "Routing in a delay tolerant network," in *Proc. of SIGCOMM*, 2004.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2472005, IEEE Transactions on Parallel and Distributed Systems



(a) Identified sub-communities. (b) Hit rate. (c) Average delay. (d) Average delay of all requests.

Fig. 9. Performance of the advanced file searching.

[5] A. Balasubramanian, B. N. Levine, and A. Venkataramani, "DTN routing as a resource allocation problem." in *Proc. of SIGCOMM*, 2007.

[6] J. Wu, M. Xiao, and L. Huang, "Homing spread: Community home-based multi-copy routing in mobile social network," in *Proc. of INFOCOM*, 2013.

[7] K. Chen and H. Shen, "Smart: Utilizing distributed social map for lightweight routing in delay tolerant networks," *IEEE/ACM Transactions on Networking*, vol. 22, no. 5, pp. 1545–1558, 2014.

[8] F. Li and J. Wu, "MOPS: Providing content-based service in disruption-tolerant networks," in *Proc. of ICDCS*, 2009.

[9] E. Yoneki, P. Hui, S. Chan, and J. Crowcroft, "A socio-aware overlay for publish/subscribe communication in delay tolerant networks," in *Proc. of MSWiM*, 2007.

[10] J. Reich and A. Chaintreau, "The age of impatience: optimal replication schemes for opportunistic networks," in *Proc. of CoNEXT*, 2009.

[11] V. Lenders, M. May, G. Karlsson, and C. Wacha, "Wireless ad hoc podcasting," *SIGMOBILE Mob. Comput. Commun. Rev.*, 2008.

[12] Y. Zhang, W. Gao, G. Cao, T. L. Porta, B. Krishnamachari, and A. Iyengar, *Social-Aware Data Diffusion in Delay Tolerant MANETs*. Springer Publisher, 2010.

[13] C. Boldrini, M. Conti, and A. Passarella, "Design and performance evaluation of contentplace, a social-aware data dissemination system for opportunistic networks." *Computer Networks*, vol. 54, no. 4, pp. 589–604, 2010.

[14] W. Gao and G. Cao, "User-centric data dissemination in disruption tolerant networks." in *Proc. of INFOCOM*, 2011.

[15] W. Gao, G. Cao, T. L. Porta, and J. Han, "On exploiting transient social contact patterns for data forwarding in delay-tolerant networks." *IEEE Trans. Mob. Comput.*, vol. 12, no. 1, 2013.

[16] X. Zhang and G. Cao, "Transient community detection and its application to data forwarding in delay tolerant networks." in *Proc. of ICNP*, 2013.

[17] N. Eagle, A. Pentland, and D. Lazer, "Inferring social network structure using mobile phone data," *PNAS*, vol. 106, no. 36, pp. 15 274–15 278, 2009.

[18] M. Mcpherson, "Birds of a feather: Homophily in social networks," *Annual Review of Sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[19] W.-J. Hsu, T. Spyropoulos, K. Psounis, and A. Helmy, "Modeling time-variant user mobility in wireless mobile networks," in *Proc. of INFOCOM*, 2007.

[20] C. E. Palazzi and A. Bujari, "Social-aware delay tolerant networking for mobile-to-mobile file sharing," *International Journal of Communication Systems*, vol. 25, no. 10, pp. 1281–1299, 2012.

[21] ——, "A delay/disruption tolerant solution for mobile-to-mobile file sharing," in *Wireless Days (WD), 2010 IFIP*, 2010, pp. 1–5.

[22] M. Pitkänen, T. Kärkkäinen, J. Greifenberg, and J. Ott, "Searching for content in mobile DTNs." in *Proc. of PerCom*, 2009.

[23] K. Chen, H. Shen, and H. Zhang, "Leveraging social networks for p2p content-based file sharing in mobile ad hoc networks." in *Proc. of MASS*, 2011.

[24] "Facebook," http://www.facebook.com/.

[25] A. Y. Halevy, "Piazza: Data management infrastructure for semantic web applications," in *Proc. of WWW*, 2003.

[26] "GENI project," http://www.geni.net/.

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

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

[29] M. Musolesi and C. Mascolo, "Designing mobility models based on social network theory." *Mobi. Comp. and Comm. Rev.*, vol. 11, no. 3, pp. 59–70, 2007.

[30] T. Repantis and V. Kalogeraki, "Data dissemination in mobile peer-to-peer networks," in *Proc. of MDM*, 2005.

[31] C. Lindemann and O. Waldhorst, "A distributed search service for peer-to-peer file sharing in mobile applications," in *Proc. of P2P*, 2002.

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

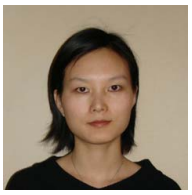
[33] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco, "Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks," *IEEE JSAC*, vol. 26, no. 5, pp. 748–760, 2008.

[34] E. M. Daly and M. Haahr, "Social network analysis for routing in disconnected delay-tolerant manets," in *Proc. of MobiHoc*, 2007.

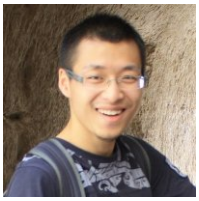
[35] A. Lindgren, A. Doria, and O. Scheln, "Probabilistic routing in intermittently connected networks." *Mobi. Comp. and Comm. Rev.*, vol. 7, no. 3, pp. 19–20, 2003.

[36] K. Chen and H. Shen, "Cont2: Social-aware content and contact based file search in delay tolerant networks," in *Proc. of ICPP*, 2013.

Kang Chen Kang Chen received the BS degree in Electronics and Information Engineering from Huazhong University of Science and Technology, China in 2005, the MS in Communication and Information Systems from the Graduate University of Chinese Academy of Sciences, China in 2008, and the Ph.D. in Computer Engineering from Clemson University in 2014. He is currently an assistant professor in the Department of Electrical and Computer Engineering at Southern Illinois University. His research interests fall in emerging networks such as vehicular networks and SDN.



Haiying Shen Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an associate professor in the Department of Electrical and Computer Engineering at 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 Li Yan received the BS degree in Information Engineering from Xi'an Jiaotong University, China in 2010, and the M.S. degree in Electrical Engineering from University of Florida in 2013. He currently is a Ph.D. student in the Department of Electrical and Computer Engineering at Clemson University, SC, United States. His research interests include wireless networks, with an emphasis on delay tolerant networks and sensor networks.