MobileCopy: Improving Data Availability and File Search Efficiency in Delay Tolerant Networks against Correlated Node Failure

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

Abstract—So far, there is no file replication method that tries to reduce data loss in correlated node failures, which however are common in Disruption Tolerant Networks (DTNs). In this paper, we propose a distributed file replication method (called MobileCopy) in DTNs, which aims to achieve low probability of totally losing a file at the expense of having a high number of impacted files in an individual large-scale correlated node failure. MobileCopy is designed for community-based file sharing systems. It has two main components: i) data loss resistant and popularity aware file replication, and ii) distributed hash table (DHT)-based file replica indexing. MobileCopy considers file popularity to determine the number of replicas of a file in each community. Through limiting the possible combination of candidate replica holders, MobileCopy greatly reduces the probability of node failures that will lead to data loss, i.e., losing all replicas of a file. Moreover, MobileCopy enables nodes to efficiently store and fetch the placement information of file replicas through competition based file replication and considering node mobility throughput among communities. Extensive trace-driven experiments demonstrate the effectiveness of MobileCopy against correlated node failures compared with previous methods.

Index Terms—File Search, Correlated Node Failure, Delay Tolerant Networks, Social Network Properties.

1 Introduction

With the rapidly growing popularity of mobile devices (e.g., smartphones, tablets, laptops), file sharing in Disruption Tolerant Networks (DTNs) has attracted significant attention. DTNs require no infrastructure and have promising applications in many scenarios such as vehicular networks, rural areas, and mountain areas, where it is infeasible or costly to build infrastructures to support data communication, but have social community characteristics that can be utilized for data communication. For example, a vehicle needs to share the information of a traffic accident that happens in a community area (e.g., photo, video) to a police vehicle far from the crash site [1], or a person using a mobile social network wants to share a newsfeed with its friend miles away [2].

However, the properties of DTNs, including network sparsity, node mobility, constrained communication range and storage space, lead to limited file availability in mobile file sharing. One simple way to increase file availability is to increase file redundancy [3]–[6]. The works in [3], [4] let individual nodes randomly and greedily replicate frequently queried files to nodes they meet. However, this method achieves high file availability at the expense of redundant resource utilization. Duong and Demeure *et al.* [6] further proposed to group nodes with stable connections and let each node check its group members' probability of requesting a file and their storage status to decide whether to create a replica. In spite of many works on file replication to enhance file availability, they fail to consider the correlated node failures [7]–[9] in DTNs, which may cause the failure of all replica nodes of a file and hence permanent file loss.

Cascaded node failure, also known as correlated node failure, refers to the scenario in which a group of nodes fail simultaneously. Many previous studies have verified that correlated node failures are very prevalent in wireless networks. Kong *et al.* [7] indicated that in wireless sensor networks constrained by limited power resource, the failure of one node can result in redistribution of communication load to nearby nodes, thereby spreading the power shutdown of some individual nodes to correlated node failures. Moreover, Xu *et al.* [8] observed that in wireless communication networks, malicious codes that originate at a small number of nodes can infect nearby mobile devices via short-range communication, leading to a "wireless epidemic" failure in a wide range of nodes. In a disaster area with a power outage, many mobile devices cannot be charged in time, which also leads to correlated node failure.

We define a *failure node set* (FNS) as a set of nodes whose simultaneous failures cause a file loss. All replica nodes of a file form an FNS. Thus, the probability of data loss of a certain file replica placement method is defined as the number of FNSs caused by the method over the total number of possible node combinations. For correlated node failures, the probability of data loss (i.e., simultaneous failures of replica nodes of a file) increases as the number of FNSs in the system increases because the probability that the failed nodes constitute one FNS increases. Since previous file replication methods use random placement for replica creation, almost every newly replicated file creates a distinct FNS, so they have a higher probability of data loss under correlated node failures. Let's assume that the number of replicas of each file is 3. For a system with N nodes, there will be $\binom{N}{3}$ node combinations that can be used to replicate a file. In the random replica placement, the probability that a file is replicated on any node combination is uniformly distributed. The simultaneous failure of nodes in a node combination can result in file loss in the correlated node failure. The distributed nature of DTNs makes it difficult for nodes to know whether a file is permanently lost. Then, the requests for the lost files would be continually forwarded in the network and congest the network, which greatly

degrades the efficiency of file sharing in DTNs.

Above examples show that when the number of files is large enough, the replication patterns of files will distribute over all possible node combinations. Andreas *et al.* [10] showed that in decentralized storage system, prevention of data loss caused by correlated node failures is preferred even at the expense of the largest data recovery cost in an individual data loss event. Therefore, our goal in this paper is to use the high number of impacted files as expense to trade for the low probability of totally losing a file. In the above example, if we replicate all files to m node combinations, the probability of file loss is reduced to $m/\binom{N}{3}$. Therefore, one effective method to reduce the probability of data loss is to limit the number of candidate replica holder combinations (m) for files, namely the number of FNSs. However, realizing such file replication in distributed DTNs is non-trivial. First, there is no central server that can help determine the candidate replica holder combinations. Second, even if these node combinations are determined and are notified to all nodes, delivering replicas to these holders will lead to long delays due to the intermittent connectivity in DTNs. Third, even if the files are successfully replicated to the node combinations, accessing the placement information of these replicas is not easy. Finally, we define file popularity as the number of queries for the file that occur during a unit period of time. It is known that creating more replicas for files with higher popularity and vice versa reduces unnecessary replicas while increasing the average file availability[1], but jointly considering file popularity and data loss reduction to enhance data availability becomes another challenge.

To handle these challenges, in this paper, we design a distributed file replication method (called MobileCopy) in DTNs, which aims to reduce the probability of data loss in correlated node failures to increase file availability. MobileCopy is specifically designed for community-based file sharing systems in DTNs such as those in [11]–[14]. The DTNs present certain social community structures, in which nodes meet a preferred group of nodes more frequently than average. Each community has a stable node as community head and nodes frequently transit to other communities as brokers for inter-community communication. MobileCopy has two main components: i) data loss resistant and popularity aware file replication, and ii) DHT-based file replica indexing.

In MobileCopy, file replication is independently conducted in individual community. First, if the querying frequency on a file in community C_i from nodes in community C_j is high, the file is replicated to community C_j. Second, each community head replicates the files in its community with the considerations of both file popularity and correlated node failures. Specifically, the community head limits the candidate replica holder combinations in its own community by grouping the community nodes. It determines the number of replicas of a file based on its popularity and launches competition between the replicas to disseminate the replicas to the nodes with meeting ability matching the file replicas' popularity.

If a node cannot find its requested file in its own community, it needs to search the file globally. We use a DHT-based file replica indexing scheme to finish the task. The application of DHT in wireless file sharing system has been widely discussed [15], [16]. Combining with the community structure extracted from nodes in our case, the replica placement information and corresponding indexing information are cooperatively maintained by representative nodes in different communities. The "index" of a file means the IDs of nodes that store the file. The DHT-based file replica indexing method maps a file to a community to store the indices of the file's replicas. Based on the DHT, a file requester can find the mapped community of the file to query the indices of the file replicas. To accelerate the speed of finding the replicas and the replica placement information, we also designed a method for setting file searching priority based on node mobility throughput, which measures the node movement intensity among communities, to help selecting the community with the most connectivity to the file request's current community. To our best knowledge, MobileCopy is the first file replication method that attempts to reduce the probability of data loss in correlated node failures in DTNs. The contributions of this paper include:

(1) We propose a novel file replication method that considers both file popularity and correlated node failures to enhance file availability.
(2) We design a distributed file replica indexing method that distributes replica placement information in communities and make them easily accessible for file searching.
(3) We conduct extensive trace-driven experiments to show the effectiveness of our file replication method and the efficiency of file searching based on the file indexing. Additionally, we also analyze the respective effectiveness of the components in improving file searching performance or resisting data loss caused by correlated node failure.

The remainder of this paper is organized as follows. Section 2 presents the detailed design of MobileCopy. Section 3 presents the experimental results of MobileCopy. Section 4 presents an overview of related work. Section 5 concludes this paper with remarks on our future work.

2 The Design of MobileCopy

2.1 Network Model and Background

We assume a DTN consisting of N nodes, denoted by n_i ($i \in [1, N]$). We also assume that each node has some available storage and is willing to store files from others to facilitate file sharing in the system. The work on how to encourage nodes to be cooperative on file replication is orthogonal to this work.

Community-based file sharing in DTNs has been studied in previous works [11]–[14]. MobileCopy is designed for these community-based file sharing systems and is built upon such a system. Like these works, MobileCopy only considers the DTNs in which nodes present community and certain mobility patterns. Nodes with high probability of meeting each other form one community. For example, in a DTN consisting of mobile devices on campus, device holders usually visit certain places, such as the library, department buildings, and dorms. As shown in Figure 1, in each community, the node with the highest centrality (i.e., stability) is chosen to be the community *head* (H),

1. It is measured by the percentage of successfully resolved requests.

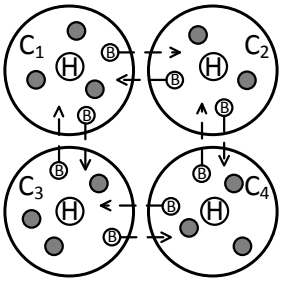


Fig. 1: Communities.

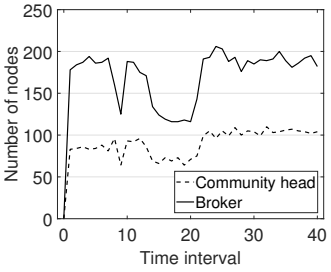


Fig. 2: Stability of the number of nodes with different roles.

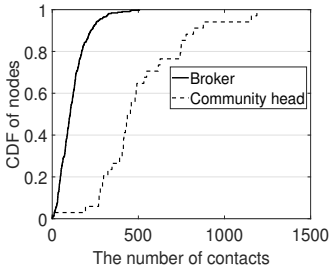


Fig. 3: Number of contacts of nodes with different roles per day.

which is used to manage the community. The node that has the highest frequency to visit each of other communities is selected as the *broker* (B) for that community. Brokers are responsible for transferring information between this community and other communities. In the case that the broker for another community cannot be found, the node that frequently visits other communities is selected as the broker since it has high mobility to other communities.

MobileCopy focuses on file replication and file replica indexing. Other issues including community detection [17], file searching [18], routing are all handled using the methods in underlying community-based file sharing system. In the following, we present the components of MobileCopy in detail.

2.2 Data Loss Resistant and Popularity Aware File Replication

2.2.1 Different Node Roles in a Community

Since nodes in one community have more contacts with each other than with nodes in other communities, and one community is a unit for file searching in previous DTN file sharing systems, MobileCopy regards each community as an autonomous subsystem for file replication. In other words, each community collects file popularity and decides replicas for each of its own files independently. Recall that the popularity of a file is defined as the number of queries of the file that occurs during a period of time. This arrangement brings about several advantages. Firstly, it is suitable for distributed DTNs in that the tasks of file popularity collection and file replication are distributed to different communities. Secondly, the file popularity can be collected more efficiently and accurately. Thirdly, since each community creates replicas based on the file popularity within its community, the created replicas can better satisfy requests of nodes in the community.

MobileCopy assigns different roles to the nodes (i.e., community head and brokers). The community head maintains the indices of all files in its community. It is also responsible for the following tasks: 1) maintaining information of the community distribution in the network; 2) conducting file replication in its community; 3) maintaining the indices for the replica placement information in its community; 4) calculating the popularity of each file in its community. The brokers are responsible for inter-community replica creation, replica placement information distribution and file searching between communities. Finally, the roles of the nodes and the community structure can be described as shown in Figure 1.

To illustrate that the number of community heads and brokers is relatively stable to fulfill their tasks, we sampled the number of nodes with each role every 12 hours in a 119-day record of wireless devices carried by students on Dartmouth College campus (called the DART trace [19]). Figure 2 shows the measured result. We see that after an initial time period, the number of nodes with each role stabilizes though with some fluctuation. The number of community heads is around 80 with a fluctuating range of [10, 15]. The number of brokers is around 200 with a fluctuating range of [10, 60]. The number of brokers fluctuates more than that of community heads. This is because that some previous brokers may change mobility patterns.

To illustrate that the nodes with different roles have sufficiently close and fast interaction to achieve the data replication task, we also measured the daily average number of contacts of the nodes with different roles in the DART trace. Figure 3 shows the measured results. We can see that although around 80% of the brokers have a daily average number of contacts fewer than 200, almost all the community heads' (more than 95%) daily average number of contacts is higher than 300. This means that most brokers may have limited interaction ability with others, but the community heads serve as the pivot of gathering and exchange information with multiple nodes. The results verify that assigning different roles to the nodes by their mobility characteristics can create sufficient node interaction opportunity and help the nodes achieve data replication in a hierarchical manner.

2.2.2 Inter-Community File Replication

The files may be shared across communities. For example, in a rural area, the nodes in a village share many farming information (e.g., humidity, insect coverage) with the nodes in another village. In order to reduce the querying delay, if the querying frequency on a file (i.e., file popularity) in community C_i from nodes in community C_j is high, the file is replicated to community C_j. Specifically, the head of each community keeps track of the querying frequency of each file in its community from every other community. If the querying frequency from community C_j is higher than a threshold, the head of C_i asks C_i's broker for C_j to carry this file to C_j when it moves to C_j. Then, the head of C_j creates replicas for the file in C_j based on its popularity in C_j. The process for inter-community file replication is illustrated in Figure 4. In this example, the head of community C_1 assigns a replica of F_1 to respective brokers for C_2, C_3 and C_4. Then, the brokers deliver the replicas of F_1 to these communities. Then, when nodes in these communities request for F_1, they can receive it from their own communities without inter-community communication.

As mentioned in the introduction section, previous file replication methods suffer from data loss under correlated node failures since they place replicas randomly on mobile

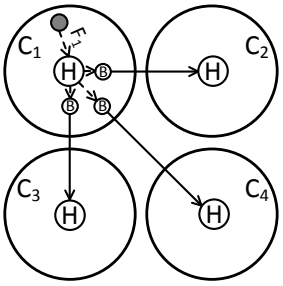
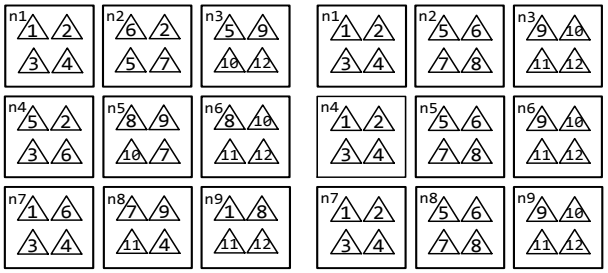


Fig. 4: Inter-community file replication.



(a) Random replication. (b) Constrained replication.

Fig. 5: Different replica placements.

nodes. In the following, we first explain why random replica placement is data loss prone to correlated node failure. Then, we introduce the solution for this issue in MobileCopy.

2.2.3 Intra-Community File Replication

In this section, we introduce how MobileCopy constrains the combination of file replica holders, utilizes file popularity to determine the number of replicas and holders, and storage limitation consideration.

Data Loss in Random Replica Placement. Randomly allocating replicas to nodes can provide strong resistance against independent node failures. As long as the size of the FNS does not exceed the number of replicas of a file, there will be at least one replica left for the file in the system. However, failures are not totally uncorrelated, and the occurrence of wide-range correlated failures is common in wireless networks [7]–[9]. Suppose the set of failure nodes include $\{n_a, n_b, n_c\}$ $(a, b, c \in [1, N])$ in a correlated failure. If a file is only replicated in these three nodes, this file is lost permanently. We use an example (shown in Figure 5(a)) to show the vulnerability of random replica placement method in correlated failures. The DTN has 9 nodes and 12 files in total. In the figure, each square stands for a node, and the triangles stand for file replicas. Suppose each file has 3 randomly allocated replicas in the system. Then, from Figure 5(a), we know that there are totally 12 FNSs used in this case. The replica placement of these files, namely FNSs, are: $File_1$: (n_1, n_7, n_9), $File_2$: (n_1, n_2, n_4), $File_3$: (n_1, n_4, n_7), $File_4$: (n_1, n_7, n_8), $File_5$: (n_2, n_3, n_4), $File_6$: (n_2, n_4, n_7), $File_7$: (n_2, n_5, n_8), $File_8$: (n_5, n_6, n_9), $File_9$: (n_3, n_5, n_8), $File_{10}$: (n_3, n_5, n_6), $File_{11}$: (n_6, n_8, n_9), $File_{12}$: (n_3, n_6, n_9). Since the total number of possible FNSs (i.e., node combinations) is $\binom{9}{3} = 84$, the probability of permanent data loss under random replica placement is $12/84$=14.3%. The expense of a data loss event is 1 file. When the number of files increases, even more FNSs will be used, leading to higher probability of the loss of some file.

We again use the setting in the example to illustrate our method (shown in Figure 5(b)). Suppose we limit the possible combinations of replica holders to only $< File_1$, $File_2$, $File_3$, $File_4 >$: (n_1, n_4, n_7), $< File_5$, $File_6$, $File_7$, $File_8 >$: (n_2, n_5, n_8) and $< File_9$, $File_{10}$, $File_{11}$, $File_{12} >$: (n_3, n_6, n_9). This means a file's replica can only be stored in any of the three combinations of nodes. As a result, only three cases can cause data loss, leading to a probability of $3/84 = 3.6\%$ for data loss, which is much smaller than the probability in the random replica placement. But the expense of data loss is higher, which is 4 files in this example. In front of pervasive correlated failure, wireless file sharing systems prefer to use the high number of impacted files to achieve the low probability of data loss event.

Limiting the combinations of replica holders aside, in file replication, file popularity also needs to be considered when creating replicas to maximize the average file availability in the system. Generally, the more popular a file is, the more replicas of the file should be created. Therefore, MobileCopy's data loss resistant and popularity aware file replication method jointly consider both objectives in file replication.

Recall that each community conducts file replication independently. Therefore, we present the file replication process in one community as an example to explain the proposed file replication method. We first introduce how to collect file popularity and how to determine the number of replicas for a file based on its popularity. Then, we present how to limit the combinations of replica holders when different files have different number of replicas.

Popularity-aware Replica Number Determination. For each file in its community, the head needs to calculate its querying frequency from its own community to determine the number of replicas of this file. Each node keeps track of the querying frequencies of its files from its own community and reports this information to the head when moving close to it. After a certain period of time, the head can know the distribution of file popularity in its community. The head node then groups all popularity values into several ranges, and each range is associated with a replica number. Then, it determines the number of replicas (denoted by R_f) for a file (denoted by F_f) based on its popularity (denoted by P_f).

Specifically, the head first determines the maximal number of replicas allowed for a file, denoted by M. We will explain how to calculate M later on. The head then sets M popularity thresholds, denoted by T_r $(r = 1, 2, ..., M)$. Then, for file F_f, its number of replicas equals $R_f = r$ if $T_r \leq P_f < T_{r+1}$, where $r \in [0, M]$. Note that T_0 and T_{M+1} are fixed to 0 and $+\infty$, respectively, to include all possible popularity values.

The values of M and T_r are determined based on the popularity distribution, the total number of files and available storage resource in the community in order to ensure that file popularities are evenly categorized into different ranges and all replicas can be stored in the community. We use a heuristic method to get these parameters. In detail, the head estimates the number of files (h_t) and the average file size (s_a) in its community. Then, since we will select M thresholds that split file popularity evenly into the $M + 1$ ranges, the amount of storage occupied by files in the $(r+1)$-th range $(r = 0, 1, ..., M)$ can be calculated as $r * \frac{h_t}{M+1} * s_a$, where r is the number of replicas for a file, $\frac{h_t}{M+1}$ is the

number of files in this range and s_a is the average size of a file for these files. We see that the files ranked higher have more replicas. Finally, M should satisfy

$$\sum_{r=0}^{M} r * \frac{N_c}{M+1} * s_a \approx \alpha S \quad (1)$$

where S is the size of available storage in the community and parameter $\alpha \in [0.5, 1)$ determines the percentage of the available storage used for storing the created replicas. A larger α will enable the file sharing system to have more replicas but causes higher storage cost on participating nodes, and vice versa. Thus, the value of α should be set according to different implementation requirements on storage availability. N_c is the total number of nodes in the community. We can deduce a suitable M based on the above equation. We then select thresholds T_r $(r = 1, 2, ..., M)$ so that the file popularity falls into all ranges evenly.

Limiting Replica Holder Combinations. After determining the number of replicas for a file, the head node selects nodes in the community to hold replicas. As indicated previously, it tries to limit the number of candidate replica holder combinations that can be selected to store the replicas of a file.

For this purpose, we can assign each node to a replica holder combination with each set having M nodes, and constrain the replicas of a file to a randomly selected combination. In this method, the most popular files will be replicated to all nodes in a combination, which may lead to load imbalance problem that some nodes become overloaded while some nodes are underloaded in the community [20]. To handle this problem, we limit the number of possible replica holders in a combination to $M + t$, where t is a small integer (e.g., 1 and 2). Then, when a file with the highest popularity is replicated to a combination, it can randomly choose M replica nodes from $M + t$ nodes. A larger t will better balance the storage load of the nodes but may reduce the storage usage efficiency, and vice versa. Thus, the value of t should be set according to different implementation requirements on storage efficiency constraint.

Specifically, MobileCopy splits N_c community nodes into $N_g = \lceil \frac{N_c}{M+t} \rceil$ groups. The replica holders of each file can only be selected from one group. Therefore, the maximal number of combinations of replica holders for file F_f is $\lceil \frac{N_c}{M+t} \rceil * \binom{M+t}{R_f}$, where $R_f \leq M$ is the number of replicas for F_f. We use an example to demonstrate the effectiveness of this proposed method. Suppose the maximal number of replicas for a file (M) is 3, t is set to 1, and the number of nodes (N_c) is 12. Nodes are split into groups with size $M + t = 4$, as shown below

$$< n_1, n_2, n_3, n_4 \| n_5, n_6, n_7, n_8 \| n_9, n_{10}, n_{11}, n_{12} >$$

Then, suppose we have a file to be replicated and the number of replicas allowed for the file is 3. The head node first randomly selects a group for the file and then randomly selects 3 nodes in the group to hold the replicas. Therefore, there are $3 * \binom{4}{3} = 12$ options to select replica holders for the file.

On the other hand, if we place replicas randomly in the community, the number of combinations of replica holders is $\binom{N_c}{R_f}$. Since N_c is often much larger than $M + t$, $\binom{N_c}{R_f} \gg \lceil \frac{N_c}{M+t} \rceil * \binom{M+t}{R_f}$. In the above example, the total number of combinations of replica holders in the random replica placement is $\binom{12}{3} = 220$, which means that the possibility of data loss is $220/12 = 18.33$ times of that in MobileCopy. Therefore, MobileCopy constrains the number of the combinations of replica holders for each file, thereby effectively preventing the probability of data loss under correlated node failures.

In real-world implementations, we can further track the failure event of each node during a certain time window (e.g., 1 week). When arranging the replica holder combinations, we always choose the nodes that never or rarely fail together during the time window to form a group. Since the options of selecting replica holders are further constrained and the arranged candidate node combinations have the lowest probability to fail simultaneously, the data loss probability can be further reduced. A larger time window size results in a better observation of the failure events of the nodes but constrains less on the selection of replica holders, and vice versa. The time window size should be set according to different implementation requirements on data loss probability upper bound and maintenance cost.

Competition based File Replica Distribution. After limiting the number of possible replica holder combinations, the probability of node failure that will cause data loss has been greatly reduced. But the combination of nodes is uniformly determined without considering the heterogeneity of nodes' capability in holding replicas and communication, namely some nodes have larger storage and may meet more nodes than the others. For the proper allocation of the file replicas, we design a competition based replica distribution scheme to differentiate the nodes' heterogeneity in sufficing file requests.

Generally, in addition to limiting the number of replica holders by the file popularity, we set different priorities for the replicas, which are jointly determined by the replica's popularity and size, so that they have to compete for the nodes with serving capability matching their priority. In MobileCopy, we use each node's meeting ability (v_k), which is measured as the weighted information entropy (diversity) [21] of the node's contact frequencies with all the other nodes that the node ever contacted (including the nodes inside and outside the node's community), to measure the node's file request service ability. More specifically, v_k is defined as:

$$v_k = -\sum_i f_{ki} p_{ki} \log p_{ki}, \quad (2)$$

where f_{ki} is node n_k's contact frequency with node n_i. p_{ki} is the ratio of f_{ki} over the sum of the contact frequencies of all the nodes that node n_k ever contacted. For example, suppose n_k's contact frequencies with three other nodes are: $f_{k1} = 20$, $f_{k2} = 10$, $f_{k3} = 5$. Thus, the ratios of each contact frequency over the sum of the contact frequencies are: $p_{k1} = \frac{4}{7}$, $p_{k2} = \frac{2}{7}$, and $p_{k3} = \frac{1}{7}$. v_k is calculated as $-20 \times \frac{4}{7} \times \log_2 \frac{4}{7} - 10 \times \frac{2}{7} \times \log_2 \frac{2}{7} - 5 \times \frac{1}{7} \times \log_2 \frac{1}{7} = 16.40$. The more nodes n_k contacted and the higher contact frequencies that the contacted nodes have, the higher v_k will be.

Each node, say n_k, periodically reports its meeting ability to the head node in its community. Then, when preparing the candidate replica holders, the community head ranks the nodes by their meeting ability in descending order, and

partitions them into $N_g = \lceil \frac{N_c}{M+t} \rceil$ groups as in Equation (3), which follows the same manner in limiting replica holder combinations:

$$< G_0 \| G_1 \| \dots \| G_{N_g-1} >$$

The lower the group id is, the higher meeting ability the group of the nodes have.

Intuitively, the higher popularity a file has, the more frequently it will be requested. Therefore, following the work of Chen *et al.* [22], we define the priority value of a file F_i as $P_i = \sqrt{q_i}$, where q_i denotes the file's popularity. According to the observations in [22], given the number of the replica holders of file F_i is R_f^i, to achieve the minimum overall file querying delay of the file, the sum of the meeting ability of the replica holders of the file (denoted as $V_i = \sum_{k=1}^{R_f^i} v_k$) should be proportional to the file's priority value, namely $V_i \propto P_i$. Since the candidate replica holders within a community are split into several groups, we also partition the popularity of the files into the same number of levels so that the files can fit in the suitable group of nodes whose capability matches the popularity of the files. Specifically, the community head first determines the maximum and the minimum values of the file priority, denoted as P_{max} and P_{min}, respectively. As all the nodes in a community are split into N_g groups, the interval of popularity between neighboring groups can be determined as

$$\widehat{P} = \frac{P_{max} - P_{min}}{N_g - 1}, \tag{3}$$

We expect that the smaller the priority level id is, the higher priority the file will have, namely the level of file priority follows the same ordering manner of the groups. Therefore, for a file with priority P_i, we define its corresponding level of priority (P_i') as in Equation (4):

$$P_i' = \lceil N_g - 1 - \frac{P_i - P_{min}}{\widehat{P}} \rceil = \lceil \frac{(N_g - 1)(P_{max} - P_i)}{P_{max} - P_{min}} \rceil, \tag{4}$$

Note that the file priority levels match the group ids (i.e., their values fall into the range $[0, N_g - 1]$). The community head will try to let each file compete for the group of replica holders whose meeting ability matches the file's priority level. For example, for the file with priority level 0, the expected group for the file is G_0. For illustration, Figure 6 demonstrates the process of the creation of a file replica in MobileCopy. Specifically, suppose node n_i needs to replicate a file F_i. The community head firstly tries to replicate F_i to the group of nodes whose meeting ability matches the file's priority. For example, suppose the priority level of F_i is P_i', it will first compete for the group of nodes whose id is equal to P_i'. If the nodes within the targeted group still have storage for creating new replicas, F_i wins the competition and proceeds to creating replicas on its targeted group of nodes. Otherwise, the file has to lower its expectation for the meeting ability of nodes and repeats the competition for the new expected group of nodes. Recall that the lower priority the file has, the less replicas it can create. Thus, the competition based file replica distribution method can finally reach a balance between node storage overhead and file popularity considering files with different popularity levels and sizes.

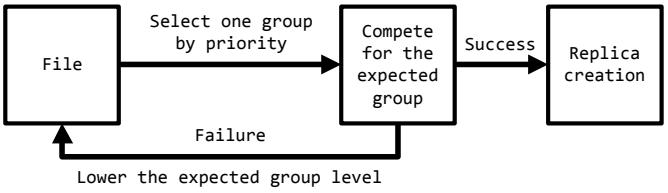


Fig. 6: Distribution of replicas.

Storage Limitation Consideration. It is possible that the storage of some node(s) in a certain group is full, which leads to the case that some extra file replicas cannot be stored at anywhere. To solve this problem, MobileCopy allows the nodes from other groups to temporarily help store the extra file replicas. Specifically, when some node(s) in a group, say G_i, have no storage space, the remaining number of nodes in the group may not satisfy the number of required replica for the file. Then MobileCopy allows G_i to borrow some nodes with available storage from neighboring groups (i.e., G_{i+1} or G_{i-1}) to temporarily store the file replicas that belong to G_i. That is, in addition to the file replicas that belong to G_{i+1} or G_{i-1}, the borrowed nodes will also store the replicas that belong to G_i.

Even with such a design, it is still possible that the storage of all nodes in a community is full. In MobileCopy, when a replica must be stored on a node without available storage, the node randomly drops some replica. Since replicas are dropped with the same probability, the ratios between the numbers of replicas of files with different popularity remain unchanged. In other words, dropping replicas randomly does not break the rule that the more popular files have more replicas.

2.3 DHT-based File Replica Indexing

2.3.1 Distribution of Replica Placement Information

Since nodes often are sparsely distributed in DTNs, it may not be easy for a requester to meet the requested file directly even though it is replicated in the network. Therefore, we need to design a scheme to efficiently maintain the placement information of each file for easy access in file searching. However, it is a non-trivial task. Firstly, a distributed method is needed to store the replicas' placement information since there is no central server or infrastructure available in DTNs. Secondly, it is desirable to distribute the placement information evenly in the network since the storage and bandwidth on each individual node is often limited. Thirdly, the placement information must be up-to-date, which means that once the replicas of a file are created, changed or deleted, the replica placement information must be updated quickly and the file requesters can always receive the correct placement information. To handle these challenges, MobileCopy uses the distributed hash table (DHT) technique to distribute the placement information of different files to different communities. DHT is well known by its balanced information distribution [23]. For each file, it is mapped to a community based on the consistency hash value [24] of its name. The mapped community is the one whose ID is equal to or follows the hash value. Then, the replica placement information of the file is stored, updated or deleted in the mapped community. Later on, when a node requests for the file, it can follow the same process to get the community that stores the file's replica placement

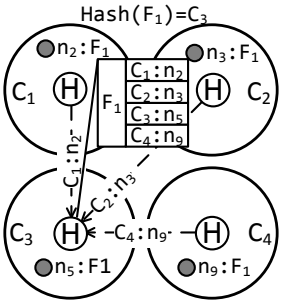


Fig. 7: Replica placement reporting.

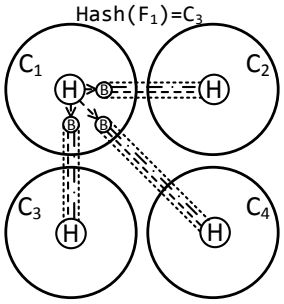


Fig. 8: Search priority based on throughput.

information. As a result, any node in the network can know where the replica placement information of its requested file is stored.

Once the replica nodes of a file are determined in a community, the head in the community generates a hash value by hashing the file name. Then, it identifies the community whose ID is equal to or follows the hash value and forwards the replica placement information to this community through the brokers. Each piece of replica placement information assigned to a community will be stored in multiple nodes that often stay in the community. This is to ensure that such information has a high probability to be found in the community even when nodes move constantly in DTNs. To select such nodes, the head node in a community first collects each community member's staying probability (i.e., the portion of time a node stays in the community during a unit time period). Then, it ranks community members in descending order of their staying probabilities and takes the top N_s nodes as the replica placement information storage nodes. N_s determines the expected probability that at least one storage node stays in the community when a file request arrives. A larger N_s leads to a larger expected probability but increases the storage overhead of the replica placement information, and vice versa. Thus, the value of N_s needs to be set according to different implementation requirements on storage usage constraint. The head node also records the IDs of these storage nodes to guide file requests in file searching. The process of replica placement information update and deletion is conducted in the same manner.

We use Figure 7 to illustrate the replica placement information distribution process. In this example, suppose F_1 is mapped to community C_3. Then, the heads in C_1, C_2 and C_4 report the replica placement information of file F_1 in their communities to C_3. Thus the overall replica placement information of F_1 is aggregated in C_3. This information is further distributed to storage nodes in C_3.

2.3.2 File Searching Priority based on Node Mobility Throughput

Generally, the nodes have different levels of mobility intensity, which is defined as the frequency of node movement during a period of time, among communities. On the one hand, the brokers, which have higher probability of transiting among communities, are primarily responsible for forwarding information between the communities, but their availability cannot always be guaranteed. On the other hand, the efficient retrieval of information is closely related with the connectivity among communities. Each time the replica placement information of the target file is obtained, there will be several communities for the request to choose from. To help selecting the community that is the most efficient for retrieving the file replica, we assign priorities for the communities based on their connectivity to the current community of the request. As shown in Figure 8, MobileCopy measures the amount of nodes moving from one community, say C_i, to another community, say C_j, to represent the connectivity from C_i to C_j. This metric indicates the data transfer capacity between each pair of communities, namely the "throughput" of communication for the cases of physical wired or wireless links. In MobileCopy, we use the term "node mobility throughput" to represent this metric, which is defined as the number of nodes that transit between two communities during a period of time (e.g., 10min). Thus, suppose the current community of the request is C_i, MobileCopy uses the node mobility throughput from C_i to each of the other communities, to determine the priority of file searching.

Since the retrieval of replica placement information and the retrieval of replicas require at least two transits (i.e., one transit to the holder, another transit back to the origin node of the request), we care about the round-trip mobility throughput between communities. For community C_i, we use S_{ij} to denote the number of nodes that move out from C_i to C_j during a period of time, and S_{ji} to denote the number of nodes that move into C_i from C_j during the same period of time. The updated round-trip mobility throughput is the average value of S_{ij} and S_{ji}. The head of the community periodically updates its throughput to community C_j by Equation (5).

$$B_{ij}^{new} = \beta B_{ij}^{old} + (1-\beta)\frac{S_{ij}+S_{ji}}{2}, \tag{5}$$

where B_{ij}^{new} and B_{ij}^{old} represent the adjusted throughput for the next unit period of time and the measured throughput for the most recent unit period of time, respectively. β is the parameter weighting the importance of the newly measured throughput and the previous throughput. A larger β means the most recently measured throughput has more impact in the adjustment but lead to less sensitive to the throughput change, and vice versa. Thus, the value of β should be set according to different implementation requirements on file search efficiency.

After the head of C_i has determined the nodes' mobility throughput to all other peer communities, it ranks the peer communities by their measured throughput in descending order, and maintains the result as shown in Table 1. Then, the process of file retrieval will be started from the community with high node mobility throughput to the community with low node mobility throughput successively until the requested file or information is retrieved.

TABLE 1: Node mobility throughput table maintained by a community head.

Community ID	Measured throughput
2	20
3	15
4	6

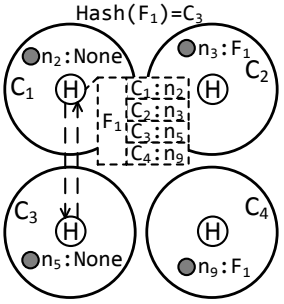


Fig. 9: Finding replica placement information.

2.3.3 Replica Placement Information Assisted File Searching

File searching can be completed efficiently in two steps. Firstly, the file requester finds the replica placement information of the requested file (i.e., ID of replica holders). Secondly, the file requester locates a replica of the requested file based on the placement information obtained.

Finding the Placement Information. This step is similar to the replica placement distribution process. When a node in community C_i requests for a file, it forwards the request to the head in the community. The head node first checks whether the replica of the file exists in some nodes in local community. If yes, the requester can learn the replica placement information directly from the head. Under the case that there is no file replica in local community (e.g., the local community only has a few requests for the file so the replicas of the file are stored in other communities), the requester will ask the head for the communities that have the file replicas. Specifically, the head node hashes the file ID to know the community that stores the file's replica placement information, say C_k. Then, the head forwards the request to the broker for C_k. After the head node in C_k receives the request, it responds the replica placement information of the requested file back to the head of C_i through brokers. The head of C_i then forwards the information to the requester. Such a process can be illustrated by Figure 9. "None" in the figure means that the node no longer has the replica. In this example, the head in the request's community C_1 firstly checks whether it has the replica placement information of file F_1 locally. If such information cannot be found in C_1, the head calculates the DHT hash value of the file, which is 3. Then, a broker carries the request to C_3. Finally, the head in C_3 responds with the replica placement information of file F_1.

Locating the Requested File. With the replica placement information of the requested file, the requester knows the holders of the file's replicas. Meanwhile, the requester also knows the node mobility throughput from the community head. Then, it can schedule searches according to the node mobility throughput to communities containing the file and the number of replicas of the file in each community. A replica holder in the community with the highest node mobility throughput has the highest priority to be selected as the target node to send the request. The requester uses the DTN routing algorithm to send the request (along with other replica holder information) to the target node. In the case that the target node no longer has the replica, the request is forwarded to other replica holders in the same community. If all other replica nodes fail though the probability is low, the forwarding process iterates.

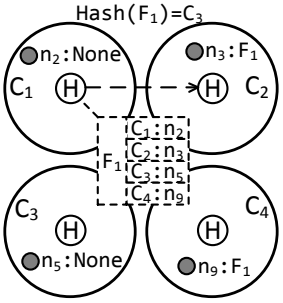


Fig. 10: Searching file.

Figure 10 shows an example on file searching. From the context of Figure 9, the requester in C_1 has known that the replicas of F_1 are originally stored by n_2 in C_1, n_3 in C_2, n_5 in C_3 and n_9 in C_4. Since n_2 in the requester's original community and n_5 in the replica placement information node's community no longer have the replica, the forwarder then transfers the request to n_3 in C_2 in the next step because it is a neighbor community and contains a replica of F_1. In this case, for files partially lost, as long as there's node holding the replica in some community, the file can be recovered. Besides, correlated failure generally happen among nodes belonging to the same community. Therefore, the total loss of data will only happen in the cases that the replica holders in all communities suffer from correlated failure simultaneously.

3 Performance Evaluation

We conducted trace-driven experiments based on the DART [19] and the DNET [25] traces. The details of the datasets are presented as follows.

Dartmouth Campus Trace (DART). DART is a 119-day record for the WLAN Access Point (AP) association activities of wireless devices carried by students on Dartmouth College campus during the period from Nov. 2, 2003 to Feb 28, 2004. To make the trace fit our analysis, we normalized the movement of the devices to buildings, and numbered the buildings as landmarks. We also filtered out nodes with few occurrences (<500) or short connections (<200s), and merged repeated records. Finally, we obtained 320 nodes and 159 landmarks.

DieselNet AP Trace (DNET). DNET is a 20-day AP association record for the movement of buses in the downtown area of UMass during the period from Oct. 22, 2007 to Nov. 16, 2007. In the collection of the DNET trace, each bus is equipped with a Diesel Brick that constantly scanned the surrounding area for available AP connections, and a GPS to record its coordinates. Since some connections are not recorded by the known APs, we filtered out nodes with few occurrences (<50) from the trace. The APs that are within certain distance (<1.5km) to each other are merged into one landmark. Finally, we obtained 34 buses and 18 landmarks. The features of the two traces are summarized in Table 2.

TABLE 2: Characteristics of mobility traces.

	DART	DNET
# Nodes	320	34
# Sub-areas	159	18
Duration	119 days	20 days
# Transits	477803	25193
# Transits per day	2401	1257

Each node originally holds 100 files for file sharing. Each file has the same size of 1KB, and each node has an available storage of 300KB. According to the analysis results of [26], the Zipf distribution with the Zipf parameter of 0.7 best models the distribution of file popularity in DHT-based file sharing systems. Therefore, in the evaluation, we set the file popularity follows the Zipf distribution with the Zipf parameter equals to 0.7. Then, the request frequency of each file was set based on its popularity, which is measured as the number of requests for the file that occur during a unit period of time. The distribution of file replicas and related indexing information needs a period of time for initialization. Since students in DART move less frequently and slower than buses in DNET, the initialization in DART needs more time than that in DNET. We set the initialization time period to 30 days for DART and 2.5 days for DNET, during which nodes collect information for file placement and receive file replicas from other nodes. After the initialization, file requests are generated based on the search rate. The search rate is defined as the number of file requests generated in each generation interval, which is 1 day for DART and 4 hours for DNET. The expiration TTL (Time-To-Live) for a file request was 4 hours and 2 hours in DART and DNET, respectively.

With each device carrier as a node, and the contact records between these nodes as the edges, of which weight is determined by the number of contacts between each pair of the nodes throughout the traces, we formulate a contact graph for DART and DNET, respectively. Through maximization of modularity [27], for DART, the nodes are partitioned into 22 communities, with each community having 15 nodes in average, and for DNET, the nodes are partitioned into 3 communities, with each community having 12 nodes in average. In determining the node mobility throughput among communities, the time period for measurement is 10min. To find the best values of t and the window size for calculating node failure probability, we vary each variable within a certain range (e.g., [1, 5] for t and [3 days, 14 days] for window size) and test different combinations of the values in the data loss experiment. Specifically, we use each combination to simulate the data loss experiment for 30 days. We find $t = 2$ and window size = 7 days is the best combination, after which the increasing rate of data loss probability significantly drops down. Similarly, we also vary the values of N_s, α and β and test different combinations of the values in the file searching experiment for 48 hours. We find the combination of $N_s = 3$, $\alpha = 0.73$ and $\beta = 0.61$ results in the highest success rate.

3.1 Data Loss Resistance

We first evaluate the data loss resistance performance of *MobileCopy* in comparison with *Random* and *Uniform*. *Random* places replicas on randomly selected nodes and follows the same method as in *MobileCopy* to determine the number of replicas for each file based on its popularity, which is the usual replica placement scheme used in mainstream wireless file sharing designs [3], [5], [22]. *Uniform* randomly chooses the replica holders and creates equal number of replicas for each file, which corresponds to methods without considering file popularity in determining the number of replicas [4].

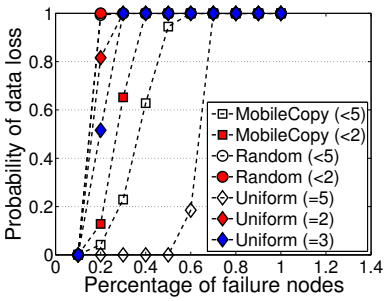


Fig. 11: Data loss probability with different node failure probabilities.

We evaluate the performance from two perspectives: data loss probability under various percentages of concurrent failure nodes and data loss probability under various sizes of nodes in a group. The percentage of failure nodes determines the scale of correlated node failures, and the number of nodes in a group determines the maximum number of combinations of replica holders for each file in a community. The percentage of failure nodes ranges from 10% to 100%, the number of nodes in each group ranges from 4 to 10, and their default values are 30% and 5, respectively. We ran the experiment for 1000 times on DART and observed data availability of each file in the end. The probability of data loss is calculated by the percent of experiments in which at least one file is lost.

In Figure 11, "($<$ x replicas)" means that a file can maximally generate x replicas in a community and "($=x$)" means that each file generates x replicas. We find that the data loss probability of *Random* is 100% when the percentage of failure nodes is larger than 20% regardless of the maximum number of replicas of a file. However, for *MobileCopy*, when the percentage of failure nodes is 20%, the data loss probability is smaller than 20%. When the percentage of failure nodes increases to 40%, the data loss probability is 60% when the maximum replica number is 5. We also find that with the increase of the maximum number of replicas, the data loss resistance of *MobileCopy* increases. This is because when more replicas are created, a file can tolerate more failure holders. For *Uniform* with 5 replicas, when the percentage of failure nodes is 60%, the data loss probability is around 20%. However, for *Uniform* with 2 replicas, when the percentage of failure nodes is 20%, the data loss probability is higher than 80%. This is because that generating more replicas for each file can significantly increase the availability of all files but the storage cost of *Uniform* with 5 replicas is 74.5% higher than that of *MobileCopy* with < 5 replicas. For fair comparison, we let *Uniform* use the approximate storage cost as that of *MobileCopy* with 5 replicas so that each file in *Uniform* can only have 3 replicas (denoted by "Uniform ($= 3$)"). From Figure 11, we can see that the data loss probability of *Uniform* with 3 replicas is only slightly lower than that of *Uniform* with 2 replicas and higher than all the results of *MobileCopy*. The result

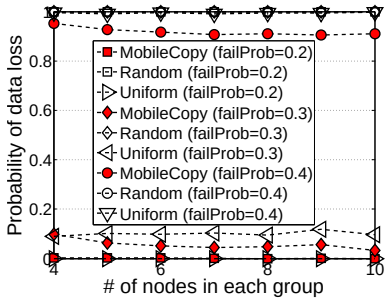


Fig. 12: Data loss probability with different # of nodes in a group.

illustrates that more replicas per file leads to higher data loss resistance but at the cost of redundant replicas for low popularity files.

In Figure 12, "failProb=x" means that the percentage of failure nodes is set to x. The curves of *Random* with different node failure probabilities overlap with each other over all node failure probabilities, which means there's always data loss under various percentages of failure nodes. We find that the data loss probability of *MobileCopy* is around 90% when failProb=0.4 and below 15% when failProb=0.2 or 0.3. We also see that with the increase of the number of nodes in each group, the data loss probability decreases under a certain node failure probabilities. This is because when the group size increases, the number of groups decreases, leading to fewer combinations of replica holders and hence lower data loss probability.

For *Uniform*, the data loss probability is below 20% when failProb=0.2 or 0.3. When failProb=0.4, the data loss probability of *Uniform* is 100%. In contrast, under these node failure probabilities, the data loss possibility of *Random* is always 100% regardless of the number of nodes in each group. Such results demonstrate the strong data loss resistance of *MobileCopy*.

3.2 File Searching Efficiency

In order to show the effectiveness of the DHT-based file replica indexing method, we further compare the file searching performance of *MobileCopy* and two representative systems: the *MOPS* publish/subscribe system [12] and the *SPOON* file sharing system [11]. To demonstrate the effectiveness of competition based file replication and node mobility throughput in increasing file searching performance, we also extend *MobileCopy* with considering file holder competition in distributing replicas and node mobility throughput in file searching, which is named as *MobileCopyAdvanced*. In MOPS, brokers from different communities exchange information about the files and the number of replicas of each file existing in their own communities when they encounter each other. In order to maximize the probability of successfully finding files, we let MOPS search the community with the maximal number of replicas of the requested file. Specifically, in file searching, a file requester forwards its request to the broker of its community (C_i), which further forwards the request to the broker of the community (C_j) that has the maximal number of replicas of the requested file. When the broker meets a file holder, it fetches the file and then forwards it back to the broker of C_i, which forwards the file to the requester. In SPOON, a file request is always forwarded to nodes that have higher meeting frequency with the target node than previous carrier. If such a node cannot be found, the request is forwarded to nodes that have higher meeting frequency with others. This method can forward a request to a file holder since nodes with similar file interests tend to gather together. For fair comparison, the total numbers of file replicas generated by all the above methods are the same.

We measured the following metrics in the experiments.

- *Success rate*: The percentage of file requests that successfully reach their target files within search TTL.
- *Average delay*: The average time (in seconds) spent by file requests to reach their target files. Note that the time spent by unsuccessful requests, which is the search TTL, is also considered in calculating this metric.
- *Average search length*: The average number of forwarding hops experienced by a file request. Note that the search length of unsuccessful requests is also considered.

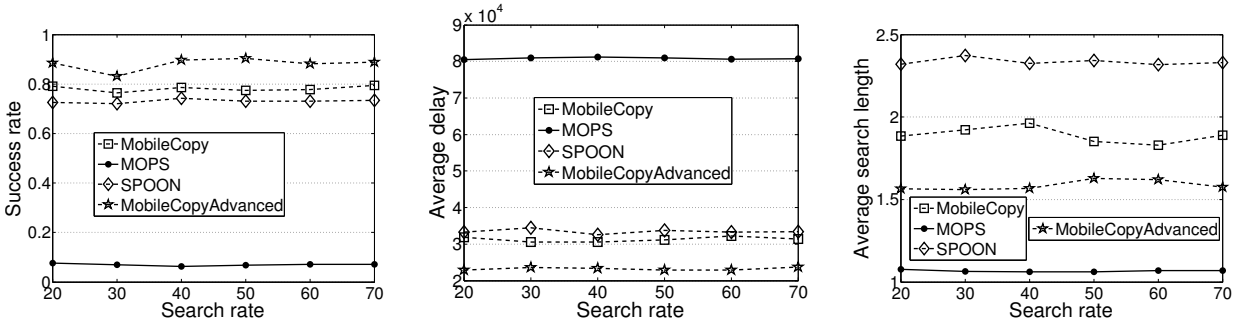
We conducted two experiments for evaluation. In the first experiment, we varied the search rate from 20 to 70. In the second experiment, we varied the search TTL of each request from 18 hours to 33 hours in DART and from 1 hour to 6 hours in DNET.

3.2.1 Success Rate

Figure 13(a) and Figure 14(a) show the success rates of the three algorithms under different search rates in DART and DNET, respectively. Figure 15(a) and Figure 16(a) show the success rates of the algorithms under different search TTLs in the experiments with DART and DNET, respectively. In these figures, we find that for both traces, the success rates follow: *MobileCopyAdvanced*>*MobileCopy*>*SPOON*≫*MOPS*. *MOPS* always has the lowest success rate under different search rates and TTLs. With the information exchanged between brokers, the community with the maximum number of the replicas of the target file (i.e., target community) can be known. However, the broker of the target community does not know the exact file holder, but can only find the file by occasional contact, rather than actively forwarding the request directly towards the target. Therefore, the request can only statically wait on the broker of the target community, which results in the lowest success rate for *MOPS*.

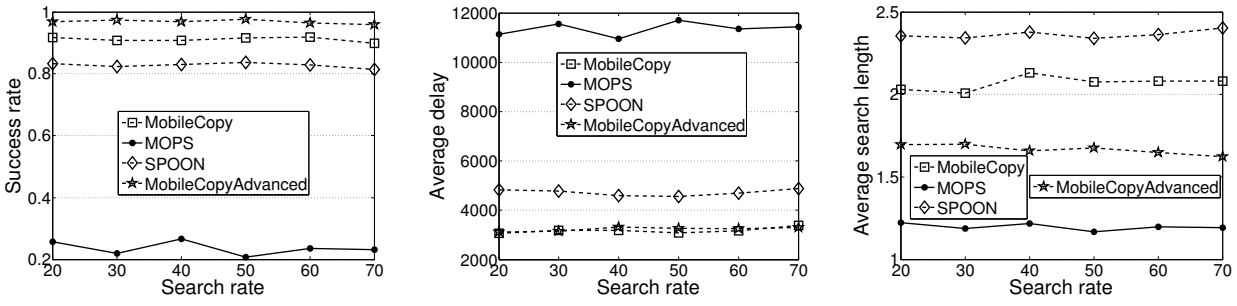
SPOON lets file requests be forwarded to nodes with higher meeting frequency with the file holders. Once attached to such a node, the request is probable to meet the replica of its target node during movement. This means that *SPOON* is likely to direct the request to the file holders through multiple contacts with their frequently met nodes. Therefore, it has higher success rate than that of *MOPS*. However, file requesters still have no clue about what nodes hold the replicas of their target file. In the case that a file request is generated in a community where few nodes met the target file holder before, *SPOON* may result in many redundant forwarding hops, which results in *SPOON*'s success rate to be lower than that of *MobileCopy*.

In *MobileCopy*, the file requester can know the holder of the file placement information through DHT at the beginning of search. Then the request is forwarded through the broker to the community and learns which nodes store the replica placement information from the head. Then,



(a) Success rate. (b) Average delay. (c) Average search length.

Fig. 13: File search performance with different search rates using the DART trace.



(a) Success rate. (b) Average delay. (c) Average search length.

Fig. 14: File search performance with different search rates using the DNET trace.

the requester knows the replica holders and schedules file searching. This is because the file replicas are distributed through node competition based on their meeting ability, consequently the request can be forwarded through the relay of nodes that frequently meet the file holder. Since in each phase of the file search process, the request has a clear target, it can reach its target file efficiently, leading to a high success rate.

In *MobileCopyAdvanced*, on the one hand, through the competition for the node storage of the expected group of nodes, the files with relatively higher popularity have been held by the nodes with higher meeting ability. On the other hand, the popular files attract more requests than the other normal files. Thus, the majority of the requests are fulfilled within their TTLs. Additionally, before the broker transferred the request, the node mobility throughput gives hints to the request which community can be reached more easily. This accelerates the delivery of the requests and the retrieval of the target files. Therefore, *MobileCopyAdvanced* built based upon *MobileCopy* achieves the highest success rate over the other methods.

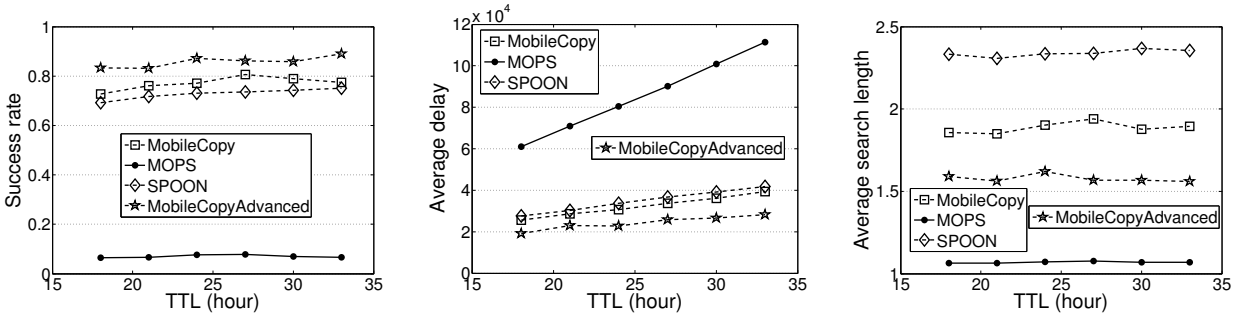
3.2.2 Average Delay

Figure 13(b) and Figure 14(b) show the average delays of the three algorithms under different search rates in DART and DNET, respectively. Figure 15(b) and Figure 16(b) show the average delays of the algorithms under different search TTLs in DART and DNET, respectively. We can see that in DART, the average delays follow: $MOPS \gg SPOON > MobileCopy > MobileCopyAdvanced$, and in DNET, the results follow: $MOPS \gg SPOON > MobileCopy \approx MobileCopyAdvanced$

As mentioned previously, *MOPS* simply forwards the request to the community with the highest number of the target file's replicas. After arrival, the request waits for the encounter with the nodes holding its target file, which consumes much waiting time. Therefore, *MOPS* generates the highest average delay. In *SPOON*, the request is forwarded to nodes that frequently meet the file holders. The request can gradually reach their target files, leading to much lower delay than that of *MOPS*. However, file requests may be generated in communities far away from the target files. As a result, they may not be able to find the nodes that can frequently meet the file holders, leading to many forwarding hops in *SPOON*. In *MobileCopy*, the request firstly identifies the community with the replica placement information, and then the details of the file replica holders. With explicit target nodes, the file request is efficiently forwarded to the target node through brokers and active nodes, resulting in the least delay. As for *MobileCopyAdvanced*, since the competition based file replication method has distributed the most frequently requested files to the frequently moving nodes, so the request has more opportunity to reach the holders of its target file, thereby reduces the time spent in file searching. Meanwhile, the node mobility throughput utilized in *MobileCopyAdvanced* can also reduce the time wasted in searching by directing the request to the community with higher mobility throughput to the request's current community. As a result, the average delay of *MobileCopyAdvanced* is lower than that of *MobileCopy*.

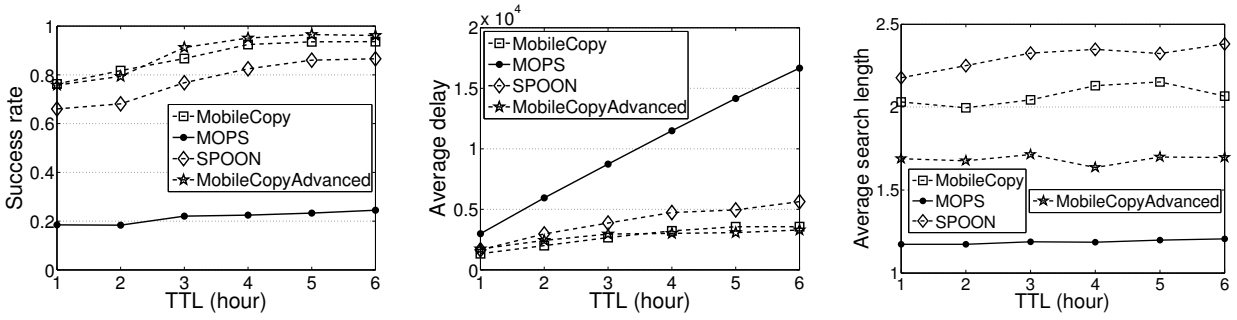
3.2.3 Average Search Length

Figure 13(c) and Figure 14(c) show the average search lengths of the three algorithms under different search rates



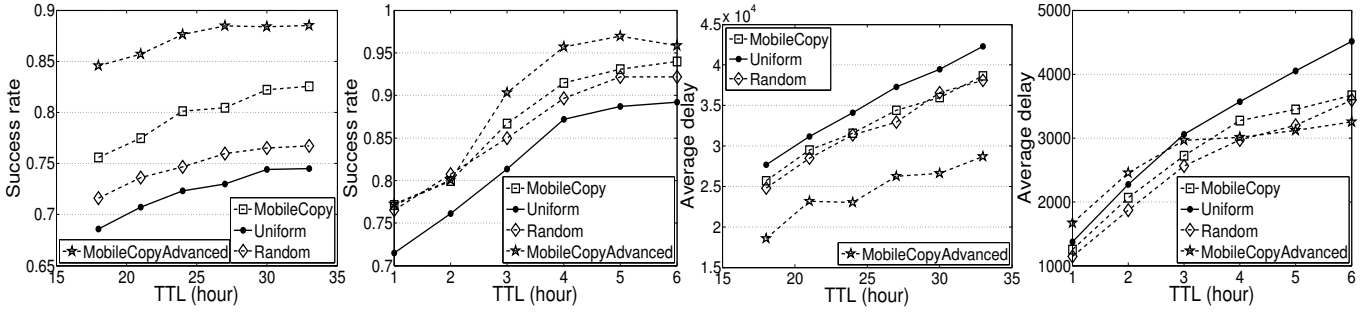
(a) Success rate. (b) Average delay. (c) Average search length.

Fig. 15: File search performance with different TTLs using the DART trace.



(a) Success rate. (b) Average delay. (c) Average search length.

Fig. 16: File search performance with different TTLs using the DNET trace.



(a) Success rate with DART trace. (b) Success rate with DNET trace. (c) Avg. delay with DART trace. (d) Avg. delay with DNET trace.

Fig. 17: Effectiveness of considering file popularity in file replication and files' competition for storage resource in improving data availability.

in DART and DNET, respectively. Figure 15(c) and Figure 16(c) show the average search lengths of the algorithms under different search TTLs in DART and DNET, respectively. We can see that for both traces, the search lengths follow: *MOPS*<*MobileCopyAdvanced*<*MobileCopy*<*SPOON*.

Recall that the search length is defined as the number of forwarding hops a file request experiences during the file search process. *MOPS* has the lowest search length since the broker passively waits for file holder, so the number of hops that the request experiences is low. In *SPOON*, since requests are not directly forwarded to the file holder, more transfer hops are needed than that of *MobileCopy*, which results in the highest search length. Since in *MobileCopy*, the requests are always proactively forwarded to the file holders, and the request has to spend some hops in finding the replica placement information of its target file through multiple hops between different communities, it has longer search length than *MOPS*. As for *MobileCopyAdvanced*, file searching based on node mobility throughput saves many transits spent on retrieving the file replica placement information and looking for the holder of the requested file. Therefore, it results in shorter search length than that of *MobileCopy* and *SPOON*. But the average search length of *MobileCopyAdvanced* is longer than that of *MOPS* in both traces. This is because the requests in *MobileCopyAdvanced* need to transit several hops to their target file holders rather than waiting for the opportunistic encounter with the file replica holder as in *MOPS*.

3.3 Data Availability

To illustrate the effectiveness of considering file popularity in file replication, and files' competition for storage resource by their popularity in improving data availability, we compare the file search performance of *MobileCopyAdvanced* and *MobileCopy* with that of *Random* and *Uniform*. In the experiment, *MobileCopy* and *MobileCopyAdvanced* can

generate maximally 5 replicas for a file per community, and 2 replicas for each file in average. Therefore, in *Uniform*, each file uniformly generates 2 replicas in each community. *Random* also generates replicas according to the popularity of files like *MobileCopy*, but randomly selects the holders. Figure 17(a) and Figure 17(b) show the success rates of the four algorithms under different TTLs in DART and DNET, respectively. Figure 17(c) and Figure 17(d) show the average delays of the four algorithms under different TTLs in DART and DNET, respectively. We can see that the success rates follow *MobileCopyAdvanced* $>$ *MobileCopy* $\approx$ *Random* $>$ *Uniform* in both traces, while the average delays follow *Uniform* $>$ *MobileCopy* $\approx$ *Random* $>$ *MobileCopyAdvanced* in DART, but follow *Uniform* $>$ *MobileCopy* $>$ *Random* $\approx$ *MobileCopyAdvanced* in DNET.

Although as indicated in Section 3.1, the data loss resistance ability of *MobileCopy* against correlated node failure is much stronger than that of *Random*, the file search performance of *MobileCopy* is similar to that of *Random*. This is because when disseminating the replicas, *MobileCopy* does not consider the heterogeneity of the nodes in contacting the others. Even if the number of replica holder for popular files is larger than that of normal files, the reachability of these file replicas is not significantly better than *Random*.

In contrast, the file search performance of *MobileCopyAdvanced* is significantly better than those of the other three methods, especially for DART. This is due to the selection of file replica holders caused by the competition for node storage resource among files, and the guidance of node mobility throughput in retrieving the target files. The results verify that *MobileCopyAdvanced* can achieve higher file availability and search efficiency than the others. The only exception is the average delays in DNET. This is because compared with DART, DNET has much fewer nodes (i.e., buses). Each node moves on its respective fixed route according to a scheduled timeline, so the meeting ability of the nodes is similar between each other, which degrades the advantage of *MobileCopyAdvanced* in distributing file replicas and directing file search. This phenomenon in turn confirms the feasibility of *MobileCopyAdvanced* for mobile networks with large number of nodes and social contact features.

3.4 Effectiveness of Components

In this section, we take a deeper analysis on the effectiveness of the respective component of MobileCopy in improving file searching performance and resisting data loss through measuring the metrics (i.e., success rate, the number of lost data) with and without using the component. In each of the following figures, the top figure is for DART and the bottom figure is for DNET.

3.4.1 Competition based File Replication

MobileCopy's competition based file replication aims to improve the file searching performance through disseminating the file replicas to nodes with proper meeting ability. To verify the effectiveness of this component, we measured the Cumulative Distribution Function (CDF) of the request success rate of all nodes with and without competition in file replication, respectively, which is shown in Figure 18. We can see that only about 30% of the nodes have request success rate higher than 80% without competition, and 50% of the nodes have request success rate higher than 80% with competition in DART, while 50% of the nodes have request success rate higher than 88% without competition, and almost 80% of the nodes have request success rate higher than 88% with competition in DNET. This is primarily because the meeting ability fully considers both the nodes' contact frequency with others and the diversity of their contacted peers, which enables the files to be distributed to the nodes that can frequently meet with various nodes. This observation confirms the effectiveness of utilizing competition based file replication in increasing file search performance.

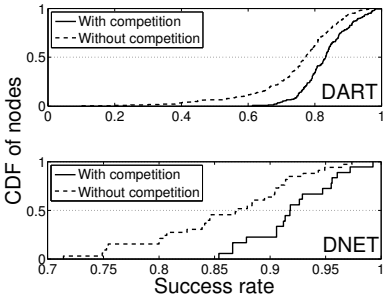


Fig. 18: Effectiveness of competition based file replication.

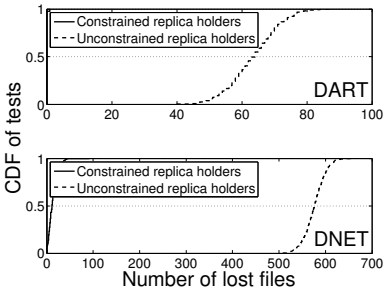


Fig. 19: Necessity of constraining replica holders.

3.4.2 Constraining File Replica Holders

Deliberately constraining the file replica holders to limited combinations of the nodes can greatly reduce the probability of node failure combinations that will cause file loss under correlated node failure. To verify the effectiveness of this component exclusively, we ran the file loss test for 1,000 times with a node failure probability of 50% in each test. Each node has 100 files to be replicated and placed in the system. Then we measured the number of lost files in each test for the case with constrained replica holders and the case with unconstrained replica holders, and drew the results as shown in Figure 19. We see that 50% of the tests suffer from more than 65 file losses with unconstrained replica holders, and less than 10% of the tests suffered from no more than 5 file losses with constrained replica holders in DART, while 50% of the tests suffer from more than 570 file losses with unconstrained replica holders, and 50% of the tests suffer from no more than 100 file losses with constrained replica holders in DNET. Obviously, constraining the combination of replica holders can greatly reduce the probability of file loss against correlated node failure.

3.4.3 File Popularity

Considering file popularity can not only help arrange the number of replicas for this file according to its request

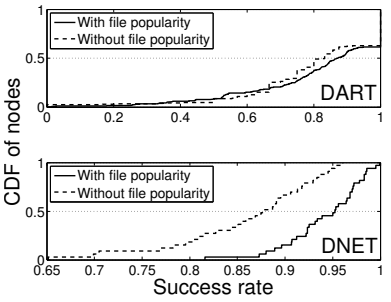


Fig. 20: Effectiveness of file popularity for file searching.

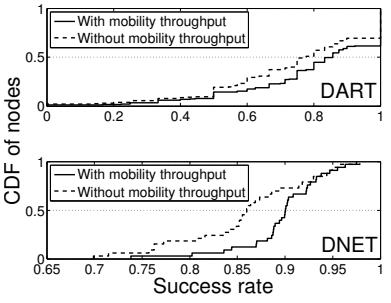


Fig. 21: Importance of mobility throughput.

frequency, but also assign the replicas to the holders with proper service ability matching the file's popularity. To demonstrate the improvement of file search performance brought by considering file popularity, we measured the CDF of the request success rate of all nodes with and without considering file popularity, respectively, which is shown in Figure 20. We see that about 50% of the nodes have success rate higher than 80% without considering file popularity and about 60% of the nodes have success rate higher than 80% with considering file popularity in DART, while about 30% of the nodes have success rate higher than 90% without considering file popularity and about 70% of the nodes have success rate higher than 90% with considering file popularity in DNET. Note the network size of DNET is much smaller than that of DART, and the buses in DNET drive on their respective determined routes by schedules, so the nodes in DNET contact more frequently and steadily than those in DART. This shows that the file popularity scoring of MobileCopy effectively enables the more frequently requested files to have more replica availability. In summary, considering file popularity in determining the number of file replicas is crucial for file retrieval performance and storage overhead balance of wireless file sharing systems, especially for wireless networks with a small size.

3.4.4 Node Mobility Throughput

MobileCopy aims to use node mobility throughput to guide the retrieval of the file replica from the community that can be most easily reached by the request. To demonstrate the effectiveness of this component, we measured the CDF of the request success rate of all nodes with and without following node mobility throughput in file searching, respectively. Figure 21 shows the measurement result. We can see that 70% of the nodes have success rate higher than 60% without considering node mobility throughput and 80% of the nodes have success rate higher than 60% with considering mobility throughput in DART, while 80% of the nodes have success rate higher than 80% without considering node mobility throughput and more than 90% of the nodes have success rate higher than 80% with considering mobility throughput in DNET. Therefore, node mobility throughput is crucial for increasing the file retrieval efficiency of wireless file sharing systems.

4 Related Work

File Replication in Distributed Wireless Networks The topic of file replication in distributed wireless networks has been extensively studied. The general method is to tradeoff the redundancy of files with resource to increase file availability efficiently. To increase the data availability in mobile ad hoc networks, Hara *et al.* [4] proposed three replica allocation methods. The methods determine the necessity of file replication or deletion based on the file access frequencies of individual nodes, neighbor nodes, and a group of stably encountering nodes, respectively. The work in [3] proposes to cache popular files on the intersection nodes on the path of requests for these files, which can be used to satisfy future file requests. Though it is effective for popular files, it fails to utilize all storage space in nodes other than the intersection nodes. Crawford *et al.* [5] proposed to estimate the evolution of file popularity by analyzing historical heterogeneous contact and request patterns of the devices, and designed a greedy algorithm based method to determine the number of files according to the estimated file popularity. Guo *et al.* [28] proposed to divide nodes into groups according to their social relationship and determine the optimal number of file replicas for each group by formulating and solving a convex optimization problem. In [6], nodes predict future file requests based on collected sematic information from their group members and create replicas accordingly. Echeverría *et al.* [29] proposed a delay-tolerant data sharing system that combines delay-tolerant protocol with a publish/subscribe mechanism to enable "many to one" information sharing. The PCS algorithm [22] considers both storage and meeting ability as resources for replica creation. It models the relationship between the average file access delay and the resource allocation to deduce how to allocate resources for files to create replicas so that the average file access delay is minimized. Chen [30] investigated how to ensure that each data can be accessed within at most k hops in mobile networks by creating file replicas. It copies a file to the right place that needs a replica so that "each data can be accessed within k hops" is still satisfied. In spite of many works on file replication in DTNs, there has been no work that tries to reduce data loss in correlated failures, which is common in wireless networks.

Popularity based File Replication and Sharing In addition, although in different scenarios, popularity based file replication and sharing has been widely explored. Liu *et al.* [31] designed a VoD system that uses coding-aware and replacement strategy using Reed-Solomon codes to enhance video availability. Liu *et al.* [32] proposed online storage systems that utilize peer assistance to reduce bandwidth and cost for online storage providers. Dai *et al.* [33] proposed a collaborative caching mechanism to achieve the traffic locality in P2P network in the context of collaborative Internet Service Provider. Liu *et al.* [34] analyzed the possibilities and trade-offs concerning server design in a hybrid

environment. Tang *et al.* [35] designed a method to boost the sync performance of personal cloud storage services by leveraging multiple available clouds to maximize the parallel transfer opportunities.

5 Conclusion

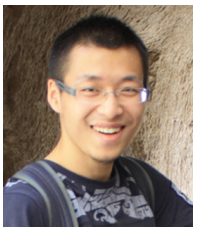
Previous file replication methods aiming at improving data availability in DTNs neglect correlated node failures, which can cause serious data loss and low data availability. We propose MobileCopy to increase the data availability in correlated node failures in DTNs. MobileCopy is designed for community-based file sharing systems and conducts file replication within each community independently. A failure node set (FNS) is a set of nodes whose simultaneous failures cause a file loss, i.e., a set of replica nodes of a file. Thus, MobileCopy aims to reduce the data loss by reducing the number of FNSs in the system. To this end, it groups community nodes and constrains the replicas of many files to the nodes in one group, and determines the number of replicas of a file based on its popularity in order to increase the average data availability for file requests. Further, MobileCopy has a DHT-based file replica indexing method, which provides an efficient way to update and fetch replica placement information of a given file. Thus, a file requester can quickly learn the replica nodes of a given file in file searching. Additionally, we improve the file searching performance of MobileCopy with competition based file replication and node mobility throughput based file searching. Extensive trace-driven experiments show that MobileCopy is robust against correlated node failures and efficient in file sharing in comparison with previous methods. Additional in-depth analysis of the components also verifies their respective effectiveness in improving file searching performance and resisting data loss in correlated node failure. For files that do not have the highest popularity, their number of replicas is less than the number of nodes in a group, which leads to several different FNSs. In our future work, we will study how to constrain the number of FNSs of these files to reduce data loss rate.

Acknowledgements

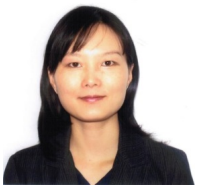
This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, Microsoft Research Faculty Fellowship 8300751.

References

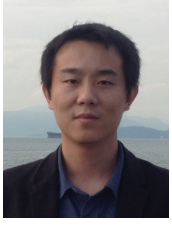
[1] M. Kim, J. Lim, H. Yu, K. Kim, Y. Kim, and S.-B. Lee, "ViewMap: Sharing private in-vehicle dashcam videos," in *Proc. of NSDI*, 2017.
[2] X. He, G. Yang, and H. Zhang, "DRANS: Daily routine analysis for node searching in delay tolerant networks," *IJDSN*, vol. 13, no. 7, 2017.
[3] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *TMC*, 2006.
[4] T. Hara and S. K. Madria, "Data replication for improving data accessibility in ad hoc networks," *TMC*, 2006.
[5] V. G. Crawford, A. Kuhnle, M. A. Alim, and M. T. Thai, "Space-efficient and dynamic caching for D2D networks of heterogeneous users," in *Proc. of MASS*, 2018.
[6] H. H. Duong and I. Demeure, "Proactive data replication using semantic information within mobility groups in manet," in *Proc. of Mobilware*, 2009.
[7] Z. Kong and E. Yeh, "Resilience to degree-dependent and cascading node failures in random geometric networks," *TIT*, 2010.
[8] Y. Xu and W. Wang, "Characterizing the spread of correlated failures in large wireless networks," in *Proc. of INFOCOM*, 2010.
[9] S. Al-Takrouri and A. V. Savkin, "A decentralized flow redistribution algorithm for avoiding cascaded failures in complex networks," *Physica A: Statistical Mechanics and its Applications*, 2013.
[10] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly durable, decentralized storage despite massive correlated failures," in *Proc. of NSDI*, 2005.
[11] 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.
[12] F. Li and J. Wu, "MOPS: Providing content-based service in disruption-tolerant networks," in *Proc. of ICDCS*, 2009.
[13] S. Lu, Y. Liu, Y. Liu, and M. Kumar, "Loop: A location based routing scheme for opportunistic networks." in *Proc. of MASS*, 2012.
[14] J. Zhao, Y. Zhu, and L. M. Ni, "Correlating mobility with social encounters: Distributed localization in sparse mobile networks," in *Proc. of MASS*, 2012.
[15] C. Cramer and T. Fuhrmann, "Proximity neighbor selection for a DHT in wireless multi-hop networks," in *Proc. of P2P*, 2005.
[16] G. Ding and B. Bhargava, "Peer-to-peer file-sharing over mobile ad hoc networks," in *Proc. of PCC*, 2004.
[17] X. Zhang and G. Cao, "Transient community detection and its application to data forwarding in delay tolerant networks." in *Proc. of ICNP*, 2013.
[18] L. Yan, H. Shen, and K. Chen, "TSearch: Target-oriented low-delay node searching in dtns with social network properties," in *Proc. of INFOCOM*, 2015.
[19] T. Henderson, D. Kotz, and I. Abyzov, "The changing usage of a mature campus-wide wireless network," in *Proc. of MobiCom*, 2004.
[20] A. Cidon, S. M. Rumble, R. Stutsman, S. Katti, J. K. Ousterhout, and M. Rosenblum, "Copysets: Reducing the frequency of data loss in cloud storage." in *Proc. of USENIX ATC*, 2013.
[21] M. Kelbert, I. Stuhl, and Y. Suhov, "Weighted entropy: basic inequalities," *Modern Stochastics: Theory and Applications*, vol. 4, no. 3, 2017.
[22] K. Chen and H. Shen, "Global optimization of file availability through replication for efficient file sharing in manets." in *Proc. of ICNP*, 2011.
[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *TON*, 2001.
[24] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web." in *Proc. of STOC*, 1997.
[25] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang, "Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing," in *Proc. of MobiCom*, 2007.
[26] J. Kangasharju, K. W. Ross, and D. A. Turner, "Optimizing file availability in peer-to-peer content distribution," in *Proc. of INFOCOM*, 2007.
[27] M. E. Newman, "Modularity and community structure in networks," *Proc. of NAS*, vol. 103, no. 23, 2006.
[28] Y. Guo, L. Duan, and R. Zhang, "Cooperative local caching and file sharing under heterogeneous file preferences," in *Proc. of ICC*, 2016.
[29] S. Echeverría, G. A. Lewis, M. Novakouski, and J. Boleng, "Delay-tolerant data sharing in tactical environments," in *Proc. of MILCOM*, 2017.
[30] X. Chen, "Data replication approaches for ad hoc wireless networks satisfying time constraints." *IJPEDS*, vol. 22, no. 3, pp. 149–161, 2007.
[31] F. Liu, S. Shen, B. Li, B. Li, and H. Jin, "Cinematic-quality vod in a P2P storage cloud: Design, implementation and measurements," *IEEE JSAC*, vol. 31, no. 9, 2013.
[32] F. Liu, Y. Sun, B. Li, B. Li, and X. Zhang, "FS2You: Peer-assisted semipersistent online hosting at a large scale," *IEEE TPDS*, vol. 21, no. 10, 2010.
[33] J. Dai, B. Li, F. Liu, B. Li, and H. Jin, "On the efficiency of collaborative caching in isp-aware p2p networks," in *Proc. of INFOCOM*, 2011.
[34] F. Liu, Y. Sun, B. Li, and B. Li, "Quota: Rationing server resources in peer-assisted online hosting systems," in *IEEE ICNP*, 2009.
[35] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo, "UniDrive: Synergize multiple consumer cloud storage services," in *Proc. of Middleware*, 2015.



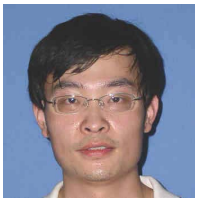
Li Yan Li Yan received the B.E. degree in Information Engineering from Xi'an Jiaotong University, China in 2010, and the M.S. degree in Electrical Engineering from University of Florida in 2013. He is currently a Ph.D. student in the Department of Computer Science at University of Virginia. His research interests include Cyber-Physical Systems and Wireless Networks, with an emphasis on Data-driven Intelligent Transportation Systems and Mobile Opportunistic Networks.



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



Kang Chen Kang Chen received the Ph.D. in Computer Engineering from Clemson University in 2014, the M.S. in Communication and Information Systems from the Graduate University of Chinese Academy of Sciences, China in 2008, and the B.S. in Electronics and Information Engineering from Huazhong University of Science and Technology, China in 2005. 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, NFV, and SDN



Guoxin Liu Guoxin Liu received the B.S. degree in BeiHang University 2006, and the M.S. degree in Institute of Software, Chinese Academy of Sciences 2009. He is currently a Ph.D. student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include distributed networks, with an emphasis on Peer-to-Peer, data center and online social networks.