MobileCopy: Resisting Correlated Node Failures to Enhance Data Availability in DTNs

Li Yan, Kang Chen, Haiying Shen and Guoxin Liu
Department of Electrical and Computer Engineering
Clemson University, Clemson, SC 29631
Email: {lyan, kangc, shenh, guoxinl}@clemson.edu

***Abstract*—Motivated by the growing popularity of mobile devices and their increasing capacities, file sharing in disruption tolerant networks (DTNs) has attracted significant attention recently. Since nodes are sparsely distributed in separated areas and are intermittently disconnected in DTNs, it is difficult to achieve high data availability in file sharing. Many previous methods enhance file availability in DTNs through file replication. However, there has been no file replication method that tries to reduce data loss in correlated node failures, which however are common in wireless networks. 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 for efficient 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.**

I. 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 battlefields, disaster areas, rural areas, and mountain areas, where it is infeasible or costly to build infrastructures to support data communication. For example, in a wireless monitoring system on battlefield, once a node captures an event as a file, other nodes may want to access the file for reference. In a disaster area, nodes need to share information such as discovered wounded persons, food and water availability and so on.

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 [1]–[5]. The works in [1]–[3] 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. To control duplicate replicas, Zheng *et al.* [4] proposed to let each node collect its neighbor nodes' file query statistics for replica creation. Duong and Demeure *et al.* [5] 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 [6]–[8] 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.* [6] 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 shut-down of some individual nodes to correlated node failures. Moreover, Xu *et al.* [7] 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. 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.* [9] 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, hence 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, it is known that creating more replicas for higher popular files 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 [10]–[13]. 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 replicates the file to nodes in one group.

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 [14], [15]. 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 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.

The remainder of this paper is organized as follows. Section II presents the detailed design of MobileCopy. Section III presents the experimental results of MobileCopy. Section IV presents an overview of related work. Section V concludes this paper with remarks on our future work.

II. The Design of MobileCopy

A. 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 [10]–[13]. 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), 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,

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

file searching, 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.

B. Data Loss Resistant and Popularity Aware File Replication

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. 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 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. The brokers are responsible for inter-community replica creation, replica placement information distribution and file searching between communities.

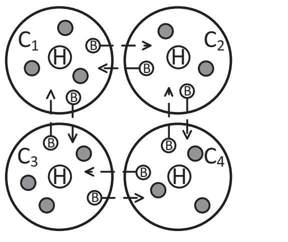


Fig. 1: Communities.

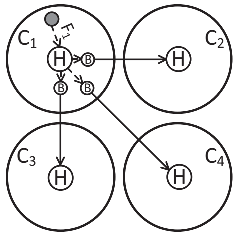
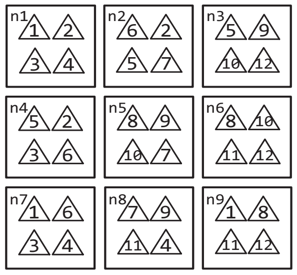
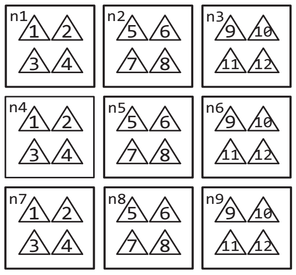


Fig. 2: Inter-community file replication.

2) Inter-Community File Replication: The files may be shared across communities. For example, in a campus, the nodes in the computer science building share many articles with the nodes in the computer engineering building. In order to reduce the querying delay, 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. 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 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 2. 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.



(a) Random replication.



(b) Constrained replication.

Fig. 3: Different replica placements.

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 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.

3) Intra-Community File Replication:

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 [6]–[8]. 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 3(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 3(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 3(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

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

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. 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 [16]. 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.

Specifically, MobileCopy splits N_c community nodes into $\lceil \frac{N_c}{M+t} \rceil$ 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.

Storage Limitation Consideration. It is possible that the storage of a selected replica node is full. To solve this problem, MobileCopy lets each node report to the head node when its storage becomes full. Then, when selecting holders for replicas, the head node excludes the nodes that do not have available storage. When a node's storage becomes available, it informs the head node so that its available storage can be fully used to store replicas.

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.

C. DHT-based File Replica Indexing

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 [17]. For each file, it is mapped to a community based on the consistency hash value [18] 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 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 is determined by the expected probability that at least one storage node stays in the community when a file request arrives. A larger expected

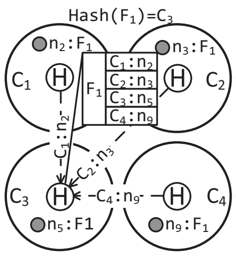


Fig. 4: Replica placement reporting.

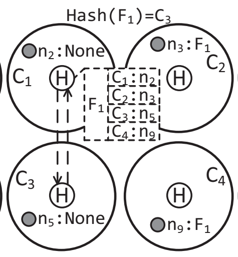


Fig. 5: Finding replica placement information.

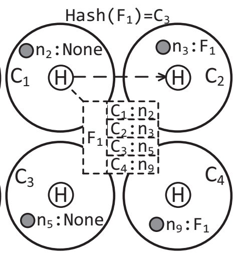


Fig. 6: Searching file.

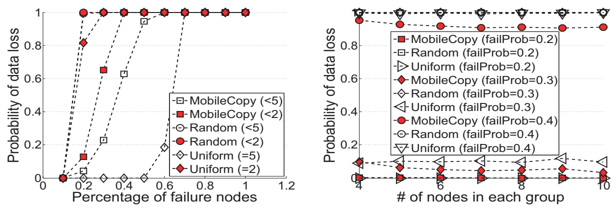
probability leads to a larger N_s and vice versa. The head node also records the IDs of these storage nodes to guide file requests in file searching. The process of replica placement information update and deletion is conducted in the same manner.

We use Figure 4 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) 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. Otherwise, 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 5. "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. Then, it can schedule searches according to the geographical distances to communities containing the file and the number of replicas of the file in each community. A replica holder in a closer community with more replicas has a higher priority to be selected as the target



(a) Data loss probability with different node failure probabilities (b) Data loss probability with different # of nodes in a group.

Fig. 7: Data loss resistance performance.

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.

Figure 6 shows an example on file searching. From the context of Figure 5, 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.

III. Performance Evaluation

We conducted trace-driven experiments based on the DART [19] and the DNET [20] traces. DART is a 119-day record for wireless devices carried by students on Dartmouth College campus. DNET is a 20-day record for WiFi nodes attached to the buses in the downtown area of UMass. We filtered out nodes with few occurrences and merged access points (APs) within short ranges to one sub-area. Finally, DART has 320 nodes and 159 sub-areas and DNET has 34 buses and 18 sub-areas. Based on trace analysis, for DART, nodes are partitioned into 22 communities, and the average number of nodes in each community is 15. For DNET, nodes are partitioned into 3 communities, and the average number of nodes in each community is 12. Each node originally holds 100 files for file sharing. Each file has the same size 1KB, and each node has 300 KB available storage. The file popularity follows a Zipf distribution with the Zipf parameter equals to 0.7 [21]. Then, the request frequency of each file was set based on its popularity. We set the initialization period to 30 days for DART and 2.5 days for DNET. After the initialization, file requests were generated based on the search rate, 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.

A. 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 [2], [4], [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 [1], [3].

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 observe 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 7(a), "($<$ 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%. The result 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 7(b), "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*.

B. 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 [11] and the *SPOON* file sharing system [10]. For fair comparison, all the three methods use the same method as in *MobileCopy* to create replicas. 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.

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. 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.

1) Success Rate: Figure 8(a) and Figure 9(a) show the success rates of the three algorithms under different search rates in DART and DNET, respectively. Figure 10(a) and Figure 11(a) show the success rates under different search TTLs in the experiments with DART and DNET, respectively. We see the success rates follow: *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 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. As a result, MOPS generates a long search delay and has low success rate.

In *MobileCopy*, the file requester can know the file holder through DHT. Then the request is forwarded through the broker to the community and learns which nodes store the replica placement information from the head. Then, the requester knows the replica holders and schedules file searching. 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 the highest success rate.

SPOON lets file requests be forwarded to nodes with higher meeting frequency with the file holders. Therefore, it has higher success rate than *MOPS*. However, a file request may be generated in a community where few nodes have met the holder of the target file, leading to many forwarding hops. Therefore, *SPOON*'s success rate is lower than *MobileCopy*.

2) Average Delay: Figure 8(b) and Figure 9(b) show the average delays of the three algorithms under different search rates in DART and DNET, respectively. Figure 10(b) and Figure 11(b) show the average delays under different search TTLs in DART and DNET, respectively. In all figures, the average delays follow: *MOPS*≫*SPOON*>*MobileCopy*.

As mentioned previously, *MOPS* simply forwards the request to the community with the highest number of target file's replicas. After arrival, the request waits for the target file. Therefore, *MOPS* generates the highest average delay. In *MobileCopy*, requests firstly identify the replica information storage nodes and the replica holders. With clear target nodes, file requests are efficiently forwarded to the target nodes through brokers and active nodes, resulting in the least delay. In *SPOON*, requests are forwarded to nodes that frequently meet the file holders. The requests can gradually reach their target files, leading to much lower delay than *MOPS*.

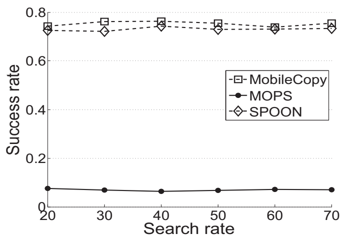
However, file requests may be generated in communities far away from the target files. As a result, they may not be able to find the nodes that can frequently meet the file holders, leading to many forwarding hops in *SPOON*.

3) Average Search Length: Figure 8(c) and Figure 9(c) show the average search lengths of the three algorithms under different search rates in DART and DNET, respectively. Figure 10(c) and Figure 11(c) show the average search lengths under different search TTLs in DART and DNET, respectively. We find that for both traces, the search lengths follow: *MOPS*<*MobileCopy*<*SPOON*.

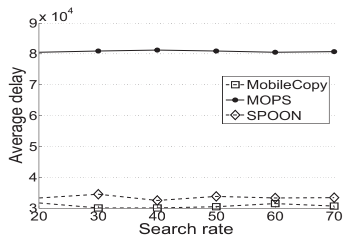
MOPS has the lowest search length since the broker passively waits for file holder. Since *MobileCopy* always actively forwards requests to file holders through multi-hops, it has higher search length than *MOPS*. In *SPOON*, since requests are not directly forwarded to file holder, more forwardings are needed than *MobileCopy*, leading to the highest search length.

C. Data Availability

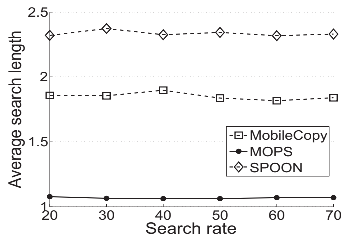
To illustrate the effectiveness of considering popularity in file replication in improving data availability, we compare *MobileCopy* with *Random* and *Uniform*. In the experiment,



(a) Success rate.

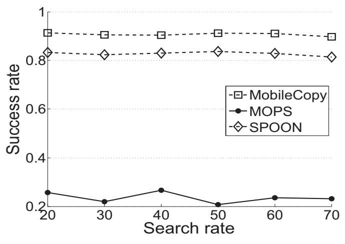


(b) Average delay.

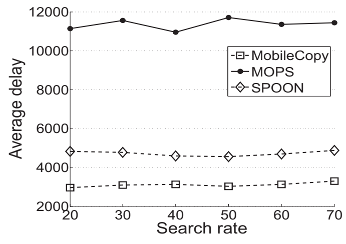


(c) Average search length.

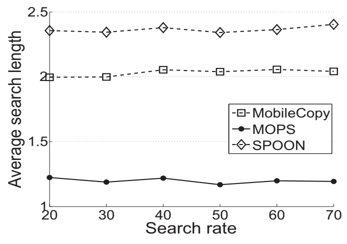
Fig. 8: File search performance with different search rates using the DART trace.



(a) Success rate.

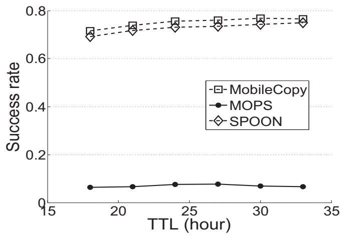


(b) Average delay.

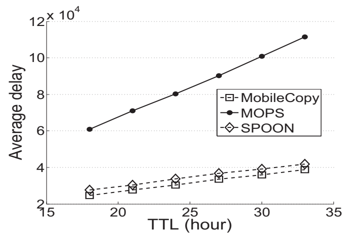


(c) Average search length.

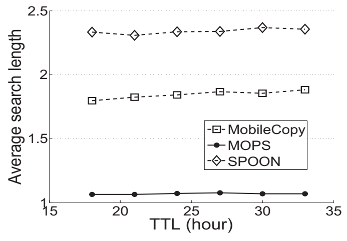
Fig. 9: File search performance with different search rates using the DNET trace.



(a) Success rate.

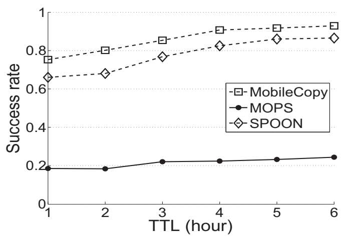


(b) Average delay.

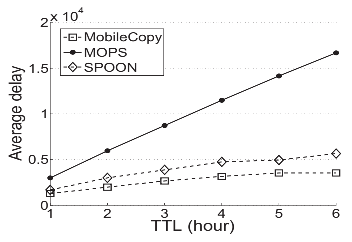


(c) Average search length.

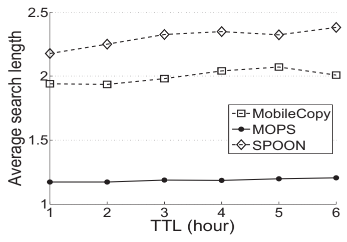
Fig. 10: File search performance with different TTLs using the DART trace.



(a) Success rate.



(b) Average delay.



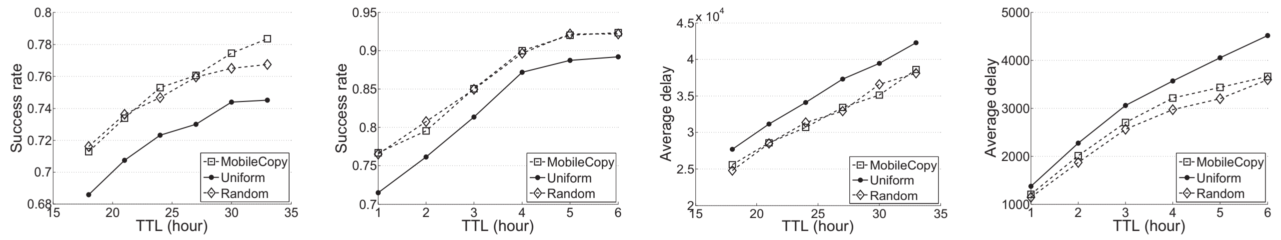
(c) Average search length.

Fig. 11: File search performance with different TTLs using the DNET trace.

MobileCopy 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 12(a) and Figure 12(b) show the success rates of the three algorithms under different TTLs in DART and DNET, respectively. Figure 12(c) and Figure 12(d) show the average delays of the three algorithms under different TTLs in DART and DNET, respectively. We find that the success rate of *MobileCopy* and *Random* is higher than *Uniform*, and the average delay of *MobileCopy* and *Random* is lower than *Uniform*. Further, the results of both metrics of *MobileCopy* are closest to those of *Random*. It is because randomly allocating replicas has no effect on success rate and average delay. The result verifies that *MobileCopy* can achieve higher file availability by considering file popularity in replica creation.

IV. Related Work

The topic of file replication in distributed wireless networks has been extensively studied. To increase the data availability in mobile ad hoc networks, Hara *et al.* [3] 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. Moussaoui *et al.* [1] proposed to realize file replication in two steps to prevent data loss. In the first step, a file is distributed in the network evenly. In the second step, nodes create replicas based on each file's density in the neighborhood to ensure the even distribution under node mobility. The work in [2] caches popular files on the intersection nodes on the path of requests for these files, which can be used to satisfy future file requests. The work in [4] creates or removes replicas on neighbor nodes based on file access requests collected from neighbor nodes.



(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. 12: Effectiveness of considering file popularity in file replication in improving data availability.

In [5], nodes predict future file requests based on collected sematic information from their group members and create replicas accordingly. 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 [23] 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.

V. 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. Extensive trace-driven experiments show that MobileCopy is robust against correlated node failures and efficient in file sharing in comparison with previous methods. 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] S. Moussaoui, M. Guerroumi, and N. Badache, "Data replication in mobile ad hoc networks," in *Proc. of MSN*, 2006.
[2] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *TMC*, 2006.
[3] T. Hara and S. K. Madria, "Data replication for improving data accessibility in ad hoc networks," *TMC*, 2006.
[4] Z. Jing, S. Jinshu, Y. Kan, and W. Yijie, "Stable neighbor based adaptive replica allocation in mobile ad hoc networks," in *Proc. of ICCS*, 2004.
[5] H. H. Duong and I. Demeure, "Proactive data replication using semantic information within mobility groups in manet," in *Proc. of Mobilware*, 2009.
[6] Z. Kong and E. Yeh, "Resilience to degree-dependent and cascading node failures in random geometric networks," *TIT*, 2010.
[7] Y. Xu and W. Wang, "Characterizing the spread of correlated failures in large wireless networks," in *Proc. of INFOCOM*, 2010.
[8] 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.
[9] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly durable, decentralized storage despite massive correlated failures," in *Proc. of NSDI*, 2005.
[10] 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.
[11] F. Li and J. Wu, "MOPS: Providing content-based service in disruption-tolerant networks," in *Proc. of ICDCS*, 2009.
[12] S. Lu, Y. Liu, Y. Liu, and M. Kumar, "Loop: A location based routing scheme for opportunistic networks." in *Proc. of MASS*, 2012.
[13] J. Zhao, Y. Zhu, and L. M. Ni, "Correlating mobility with social encounters: Distributed localization in sparse mobile networks," in *Proc. of MASS*, 2012.
[14] C. Cramer and T. Fuhrmann, "Proximity neighbor selection for a DHT in wireless multi-hop networks," in *Proc. of P2P*, 2005.
[15] G. Ding and B. Bhargava, "Peer-to-peer file-sharing over mobile ad hoc networks," in *Proc. of PCC*, 2004.
[16] 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.
[17] 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.
[18] 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.
[19] T. Henderson, D. Kotz, and I. Abyzov, "The changing usage of a mature campus-wide wireless network," in *Proc. of MobiCom*, 2004.
[20] 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.
[21] J. Kangasharju, K. W. Ross, and D. A. Turner, "Optimizing file availability in peer-to-peer content distribution," in *Proc. of INFOCOM*, 2007.
[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] X. Chen, "Data replication approaches for ad hoc wireless networks satisfying time constraints." *IJPEDS*, vol. 22, no. 3, pp. 149–161, 2007.