FILE SHARING ON A MOBILE AD HOC NETWORK

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

For the Degree of Master of Science

in the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

by

Lixin Wang

Spring 2003

Permission To Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9

Abstract

Data sharing, as a basic form of information sharing, is one of the most important reasons for the existence of networks. File sharing, as one form of data sharing, is an important issue on mobile ad hoc networks. Although much research has been done on mobile ad hoc networks, file sharing has not been well studied. Without file sharing support, the advantages of mobile ad hoc networks are limited.

The characteristics of a mobile ad hoc network pose many challenges for file sharing support. The peer-to-peer nature of a mobile ad hoc network makes server-based traditional distributed file systems unsuitable for mobile ad hoc networks. The frequent and unexpected network partitioning causes some files to be inaccessible, decreases the overall file accessibility, and makes it difficult to keep files consistent in the presence of file updates.

With its resiliency to network failures and high data availability, optimistic replication has become one of the enabling techniques for mobile computing. In this study, optimistic replication is used in combination with a peer-to-peer model to support file sharing, including read and write, among mobile host on a mobile ad hoc network. Three on-demand replication approaches are proposed as replica management policies, and their performance is examined through simulation experiments. Two reconciliation policies are also proposed to maintain replica consistency in the presence of file updates using pairwise reconciliation processes. Their performance is investigated as well. In addition, as a part of replica management, four different replica replacement policies are examined. The results show that the LRU replacement policy is a good choice on a mobile ad hoc network, and it is feasible to use the proposed approaches to support file sharing, including read and write, among mobile hosts on a mobile ad hoc network under certain situations.

Acknowledgements

I am indebted to many people who have contributed in one way or the other to the successful completion of my Master's programme. First and foremost, I would like to thank my supervisor, Prof. Rick Bunt, for his constant support, guidance, valuable suggestions, and encouragement throughout my research work, for providing me the funding, and for the many hours he devoted to discussing and reading this thesis. I would also like to thank my thesis committee members, Ralph Deters, Dwight Makaroff, and Janelle Harms, the External Examiner, for their valuable comments and suggestions.

Special thanks go to the Telecommunication Research Laboratories (TRLabs) for providing the funding, as well as a very conducive working environment. Thanks to Jack Hanson, Vera Ljubovic, and other people in TRLabs, all of whom made my stay there enjoyable and rewarding.

Thanks to Jan Thompson and other people in the department for providing necessary assistance on numerous occasions. Thanks especially go to my colleague, Wenguang wang, for sharing his knowledge on computer science with me.

In particular, I would like to thank my brother-in-law, Dr. Litong Zhao, for his valuable suggestions on writing the thesis.

Finally, I am deeply grateful to my wife, Yimei. Without her incredible support and encouragement, it would have been impossible for me to finish this thesis.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AFS	Andrew File System
AF-DWF	Access Frequency combined with DWF replica management policy
AF-LAF	Access Frequency combined with LAF replica management policy
AF-LFU	Access Frequency combined with LFU replica management policy
AF-LRU	Access Frequency combined with LRU replica management policy
CH	Closest Host reconciliation policy
DAFN	Dynamic Access Frequency and Neighbourhood
DCG	Dynamic Connectivity-based Grouping
DWF	Dynamic Write Frequency replacement policy
FO	File Owner reconciliation policy
FS-DWF	Fast-Spread combined with DWF replica management policy
FS-LAF	Fast-Spread combined with LAF replica management policy
FS-LFU	Fast-Spread combined with LFU replica management policy
FS-LRU	Fast-Spread combined with LRU replica management policy
LAF	Lowest Access Frequency replacement policy
LAN	Local Area Network
LFU	Least Frequently Used replacement policy
LRU	Least Recently Used replacement policy
Manet	Mobile Ad hoc Network
NFS	Network File System
SAF	Static Access Frequency
WAN	Wide Area Network

Chapter 1

Introduction

Data sharing, as a basic form of information sharing, is one of the most important reasons for the existence of networks. It is well supported by different systems, such as file systems, database systems, the Web, and various forms of groupware, on different networks, such as LANs, WANs, and the Internet. The strategies used to support data sharing and their performance depend heavily on the characteristics of networks. For example, on a well connected LAN, a traditional file system, such as NFS [54] or AFS [32], is able to provide sufficient support for file sharing, as one form of data sharing, among local users. To support mobile users on the Internet for file sharing, data replication (addressed in details in Section 2.1) is adopted by file systems to improve their performance [27, 37].

Recent advances in radio communication and computer technologies have led to an increasing interest in mobile ad hoc networks. File sharing support is an important issue there, but it has not been well studied. Without file sharing support, the advantages of mobile ad hoc networks are greatly reduced.

A mobile ad hoc network (Manet) is an autonomous, self-organized system composed of mobile nodes connected by wireless links [20]. Nodes are free to move and organize themselves without the restrictions of any fixed infrastructure, such as centralized access points or base stations. In such a network, each node acts both as a host and as a router. Due to the limited transmission range of wireless network interfaces, wireless communications between a pair of nodes could be single-hop communications, if the two nodes are within radio communication range, or multi-hop communications, if the two nodes are far away from each other so that the communications rely on one or more intermediate node(s) as relay(s). Quite often, several nodes may need to cooperate with each other to carry out a certain task.

Mobile ad hoc networks differ significantly from wired networks in terms of network topology, network configuration, and network resources. The main characteristics of a mobile ad hoc network can be summarized as:

- The topology of the network may change rapidly and unpredictably due to the nodes moving.
- The network can get partitioned and reconnected over time, and the wireless links between nodes may not be reliable.
- Limited resources are available, such as limited communication bandwidth, limited battery power, limited storage capacity, and so on.

Despite their limitations compared to wired networks, mobile ad hoc networks have two attractive advantages: 1) a mobile ad hoc network can be quickly deployed since it does not need any fixed infrastructure; and 2) a mobile ad hoc network is very robust in the face of single-point failures since all nodes in the network are equal in status.

Situations that need such features include establishing survivable, efficient, dynamic communication for emergency/rescue operation, disaster relief efforts, and on-the-fly conferencing. In such situations, it is either impossible or too expensive to deploy a wired network.

File sharing is one of the important functionalities that should be supported on a mobile ad hoc network. As an example, in a mobile ad hoc network for a disaster relief effort, a team member can share his/her progress report with the others so that the other members can act accordingly. In addition, the report will change after a time period. Hence file consistency should be maintained in a timely manner so that team members can get up-to-date reports. As another example, in a conference situation, a mobile ad hoc network could be set up to avoid a temporary expensive wired network. Attendees can share their files on the network. For example, they may want to read the others' presentation files in advance. Once files are updated, the modified versions should be propagated through the network to keep consistency of files. Therefore, without file sharing support, the functionalities of a mobile ad hoc network are limited.

Research in the area of mobile ad hoc networking is receiving much attention from academia, industry, and government. However, it is still far away from maturity and there are difficulties in making a mobile ad hoc network into a real working system. Most of the research effort associated with mobile ad hoc networks has focused on routing protocols [52, 47, 33, 13, 36, 56, 8], and security issues [41, 61, 7]. Routing protocols allow a host to discover routes to its communication target(s) efficiently. Security issues address authentication services to prevent intrusions of malicious mobile hosts. Little has been done in the application layer or with respect to the degree to which an application can tolerate network partitions and slow wireless links, such as supporting file sharing among mobile hosts. The purpose of this thesis is to study the issues related to file sharing support on a mobile ad hoc network in order to provide a range of options for building a file system to support file sharing among mobile hosts.

1.1 Problem Description

The characteristics of a mobile ad hoc network pose many challenges for file sharing support. There are no central server machines in a mobile ad hoc network, and mobile nodes in the network are free moving peers. Thus, server-based solutions, such as traditional file systems NFS [54] and AFS [32], don't suit mobile ad hoc networks. In particular, a mobile ad hoc network may frequently and unexpectedly get partitioned over time. Network partitioning causes some files to be inaccessible during some time periods, decreases the overall file accessibility, and makes it difficult to keep files consistent in the presence of file updates.

To alleviate the problems caused by poor network connectivity and network partitioning, data replication (addressed in Chapter 2) plays an important role. A significant amount of work has been done in developing new replication strategies to provide various services. File systems have been successfully adapted to provide mobile file access for mobile clients by using replication techniques [21, 30, 37]. Database systems adopt replication techniques to minimize query latency [43, 35, 11]. More recently, peer-to-peer file sharing systems [14, 45] use replication techniques to improve file availability.

Data replication can also be used to support file sharing among mobile hosts on mobile ad hoc networks. However, different situations require different data replication strategies emphasizing different aspects. The special characteristics make a mobile ad hoc network significantly different from a conventional wired network. Thus a fundamentally different replication strategy is required.

Two important issues in supporting file sharing on a mobile ad hoc network using data replication techniques must be addressed: one is *replica management*, which decides when to create a replica, where to place a replica, and when to destroy a replica, according to the replica management policy used. The other is *replica reconciliation*, which decides when and where to propagate file updates according to the replica reconciliation policy used. The necessity can be seen from the following points:

- The peer-to-peer nature of a mobile ad hoc network requires mobile hosts in the network to assist each other. Hence a sophisticated replica management approach is required in a mobile ad hoc network. While a mobile host creates file replicas to maximize the benefit to itself, it must also consider how to assist the others with the replicas in its cache. This is different from a mobile file system [21, 30, 37] supporting disconnected operation [37], where a mobile host creates file replicas used only by itself during a disconnected period.

- The dynamic network topology of a mobile ad hoc network requires a dynamic replication strategy. In a wired network, some long-term “spots” can always be good choices to place server machines for data replication to provide high performance according to the network topologies. In a mobile ad hoc network, the network topology changes all the time. Therefore, there are no ideal mobile hosts to allocate file replicas to for a long time to achieve overall optimized performance. This requires that replica management in a mobile ad hoc network should be able to dynamically adjust itself to deal with the dynamically changing network topology.

- It is difficult to maintain replica consistency on a mobile ad hoc network in the presence of file updates. In client-server model replications [21, 30, 37, 43], a file replica on the client side is reconciled with the server file to propagate file updates. In a mobile ad hoc network, however, the lack of central control combining with frequent network partitioning makes it difficult for a host to decide which host should be chosen as a reconciliation partner to propagate file updates. In addition, to improve overall file accessibility, it is desirable to create as many replicas as the cache space of a mobile host allows. But more file replicas make replica consistency more difficult to maintain in the presence of file updates.

1.2 Thesis Overview

In this study, optimistic data replication is combined with a peer-to-peer model (addressed in Section 2.1) to support file sharing among mobile hosts on an ad hoc network. An environment is assumed where each host has limited cache space to create file replicas. Files are allowed to be updated at any time by any host. Replica consistency is kept by using periodical pairwise reconciliation processes (see Section 2.3 for details). Attention is paid to replica management and replica reconciliation. Three on-demand replica management policies, and two reconciliation policies are proposed to support file sharing among mobile hosts by alleviating the problem of network partitioning. A series of simulation experiments were conducted to investigate the performance of the proposed policies. In addition, as a part of replica management, four replica replacement policies[1] are proposed. Their performance was also examined by simulation experiments.

The reminder of the thesis is organized as follows. Chapter 2 provides background information on data replication techniques used to alleviate the problems associated with unreliable connections and network partitions. Related work on data replication is also addressed. In Chapter 3, the system model and its operation are described. The proposed replica management policies, replica replacement policies, and reconciliation policies are also described in detail. In addition, the performance metrics which are used in the performance evaluation of the proposed policies, and factors which affect their performance, are identified. Chapter 4 describes the synthetic workloads used in simulation experiments, and the simulation model along with the general parameter settings. The simulation results for the performance evaluation of the five replica management policies, four replica replacement policies, and two reconciliation policies, are presented and discussed in Chapters 5, 6, and 7, respectively. In Chapter 8, the conclusions from this study and some possible directions for future work are presented.

[1]Since each mobile host has limited cache space, replica replacement occurs when the cache is full and a new file replica must be brought in.

Chapter 2

Data Replication

In order to provide data service on a mobile ad hoc network, a system needs to overcome the problems associated with mobile ad hoc networks caused by frequent network partitioning and low bandwidth wireless links. These two factors also exist in current wide area networks (WAN). For example, host failures may cause WAN to partition. To solve these problems, data replication plays an important role. It has been studied extensively in the areas of distributed database systems, distributed file systems, and more recently, peer-to-peer file sharing systems. The two main advantages of data replication are:

- Locating replicas of data close to the location where they will likely be used can minimize the impact of network latency.
- The existence of multiple replicas of data can improve data availability and reliability in the face of network or server failures.

In general, data replication is an effective way to improve both the performance and the availability of a distributed system in which the network connectivity is not always reliable, and the network can get partitioned sometimes. Therefore, data replication provides a way to support data services on mobile ad hoc networks.

This chapter provides background information about data replication services, upon which this study is based, and then discusses previous work on data replication techniques.

2.1 Issues in Data Replication

Approaches to data replication can be divided into two categories, pessimistic replication and optimistic replication [16], depending on the controls used to achieve global consistency across all the replicas.

2.1.1 Pessimistic Replication

Traditional pessimistic replication offers *single-copy semantics*, which gives the user the illusion of having only a single, highly available copy of a data item by keeping the replicas consistent all the time through employing some form of quorum consensus [31], voting [22], tokens [42], or primary-site [46] techniques to prevent conflicting updates. Pessimistic replication always makes worst case assumptions. If an operation on a replica can cause update conflicts among the replicas, the operation is not allowed to access the replica. Therefore, pessimistic replication algorithms prevent conflict before it occurs. If one of the replicas fails, the whole system may halt. As the number of data replicas increases, the probability of one-replica failure increases, and thus the data availability decreases. Pessimistic replication is essential in applications such as banking that must avoid giving a wrong answer. Pessimistic replication can be very expensive, however, because of the stringent hardware requirements. For example, the primary-copy with fail-over algorithm [6], a popular pessimistic replication algorithm, assumes that it can accurately distinguish a crashed node from a live but unresponsive node (i.e., caused by link failure or heavy load) so that it can reliably re-elect a primary replica should it fail. But such distinction is theoretically impossible [18, 12]. It can be probabilistically possible by investing in highly reliable network hardware and computers so that an unresponsive node can be considered as a crashed node.

2.1.2 Optimistic Replication

Optimistic replication, by contrast, allows inconsistency among the replicas, but in a controlled way. It propagates updates in the background and offers *single-copy* *availability*, which allows any read or write operation to be performed on a file as long as any one of the replicas of the file is accessible. This feature makes optimistic replication more efficient in the presence of unreliable network media. In general, optimistic replication assumes that concurrent updates, or conflicts, are rare and most conflicts can be resolved transparently without user involvement. Optimistic replication greatly improves data availability, with the risk of serving stale data to applications. Optimistic replication is used to improve the performance and availability of data service in a wide-area distributed environment. In these situations, communication between sites is slow and unreliable. Thus optimistic replication is a key enabling technology in mobile computing [19], since mobile hosts need to locally replicate all the data they need to keep working while they are disconnected, and merge the modifications with other replicas when they are reconnected. Coda [55], for example, uses optimistic replication to support disconnected operation [37].

2.1.3 The Advantages of Optimistic Replication

Although there are some costs to using optimistic replication, such as a read returning stale data by reading an out-of-date replica and concurrent updates producing conflicting versions of the same file, in general, there are some advantages of optimistic replication over pessimistic replication [53]:

- **Fault tolerance:** Optimistic replication works over slow or unreliable network links because it propagates updates among replicas in the background. Moreover, it allows users to read or update an object as long as one replica is available.

- **Networking flexibility:** In many cases, optimistic replication works over intermittent links by allowing updates to be exchanged between any pair of hosts.

- **Low cost:** Because of its networking flexibility, optimistic replication can be deployed using inexpensive hardware compared to pessimistic replication.

- **Site autonomy:** Optimistic replication improves site autonomy by requiring less coordination among sites. For example, some services (e.g., FTP mirroring) allow a replica to be added without any administrative change on the existing replicas.

2.2 Replication Models

A replication model refers to the relationship between replicas. It can be used with pessimistic replication, optimistic replication, or a combination of both.

There are three basic replication models: *master-slave, client-server, and peer-to-peer*. Each is discussed with respect to their communication, synchronization, and update generation characteristics.

2.2.1 Master-Slave Model

The master-slave model assigns one replica as *master*, while all other replicas are *slaves*. The requirement is that slaves should always be identical to the master. The model is very simple and yields a simple implementation. However, it offers limited functionality: the slaves are essentially read-only. Most master-slave services ignore all updates or modifications performed at the slave, and *undo* the update during synchronization, making the slave identical to the master. In general, modification can be performed only at the master, and slaves must synchronize directly with the master.

2.2.2 Client-Server Model

The client-server model is similar to the master-slave model in that it designates one server which serves multiple clients. However, the functionality of the clients is greatly improved, and multiple inter-communicating servers are permitted. Servers periodically synchronize with each other to make all server replicas consistent. All types of data modification and updates can be generated at the client. The client-server model has been successfully implemented in replication systems such as Coda [55].

The major drawback from the mobility point of view is the communication restriction placed on the clients. Clients can communicate with one of the servers, but they can not directly synchronize with each other. This limits their functionality when clients are in the mobile state. For example, two clients in the same room can not directly exchange updates; they must both communicate with the remote servers, as one uploads its changes and the other downloads them.

Although multiple servers make client-server models more scalable and more robust in the presence of single server failure, the absence of communication between client replicas limits the advantages provided by data replication algorithms.

2.2.3 Peer-to-Peer Model

The peer-to-peer model removes the restriction of client-server replication. The peer model is not class-based: all replicas are equals, or peers. It permits any replica of a data item to exchange update information with any other replicas. This feature allows *pairwise* communication between peers [28] and makes all peers as autonomous as possible. Thus update information can be propagated to all peers without the requirement that all peers are connected at the same instant, and therefore, single point of failure is removed automatically. The peer-to-peer model is quite useful to deal with poor connectivity, and has been implemented in systems such as Bayou [57], Ficus [27], and Rumor [28].

The peer-to-peer model provides a very rich and robust communication framework. However, it has typically suffered from scaling problems. Peer-to-peer models have traditionally been implemented by storing all necessary replication knowledge at every site, each replica thereby has full knowledge about everyone else, and synchronization and communication are permissible between any two hosts. Such an approach results in large replicated data structures (e.g., lists that contain the set of all known replicas), and thus does not scale well. Additionally, distributed algorithms that determine global state must communicate with each replica at least once and often twice. This leads to scaling problems too.

2.3 Consistency Issues

The central issue of replicated data service is how to maintain consistency between all the replicas of a data item, in other words, how to deal with updates. Pessimistic replication prevents replicas of a data item from becoming inconsistent by various methods. Primary copy, majority consensus, weighted voting and quorum consensus are all variants of the same strategy. They allow updates to occur in at most one connected environment, then propagate those updates to other replicas when connection is re-established. Therefore, maintaining replica consistency is a trivial task in pessimistic replication services.

Optimistic replication, by definition, can not guarantee a strict *single-copy* consistency. It allows updates to be committed at any replica without knowledge of other replicas, and allows conflicts to occur. Thus it relies on underlying consistency protocols, such as reconciliation [28] or anti-entropy [48], which are processes to bring two replicas into a consistent state to maintain replica consistency. In general, there are two types of consistency:

- **Eventual consistency:** Eventual consistency guarantees replica consistency in a quiescent environment. In other words, it guarantees that whatever the current state of the replicas, if no new update is issued, no replica fails, and the replicas are able to communicate freely for a long enough period, all the replicas' contents become identical.

  How to achieve eventual consistency in the presence of concurrent updates is a design choice based on the applications being supported. For example, keeping track of the order of updates is much more important in database systems than in general filing systems.

- **View consistency:** The fact that replica's contents will eventually converge says nothing about the quality of data since a user may get different versions of a replica in any order. The goal of view consistency is to control the quality of data by intervening on a replica read request (and sometimes on an update) in order to improve user-perceived consistency. This guarantee is optional, and in fact, many systems serve arbitrarily stale data to users.

To maintain replica consistency, the underlying consistency protocols need to decide how to propagate updates and how to detect and resolve conflicts among updates. These issues are addressed in the following sections, focusing on the peer-to-peer model with optimistic replication.

2.3.1 Propagating Updates

Most systems use a one-way update propagation method, either pull-based or push-based, which allows one replica to bring another up-to-date. The main advantage of one-way propagation is that it is easy to support using physical media, such as a disk, to propagate updates when a network connection is not available. Of course, two-way update propagation has its own merits. It can speed up the update propagating process since in one synchronizing process, two replicas can get newer information from each other.

Updates can be propagated in two forms, log-transfer or content-transfer. Pull-based log-transfer update propagation is often used in replicated database systems, such as Bayou [48] and Deno [11], since updates in database systems are easily logged into a file. A basic algorithm is to let each replica log all the received updates, along with timestamps associated with each update at the moment they are issued. The log files are stored on disks to survive temporary node failures. During reconciliation processes, the necessary update logs in the log files are sent to peer replicas either periodically or on demand. Given enough time for replicas to communicate with one another at least indirectly, all replicas will receive all the updates performed on all the replicas.

Timestamp vectors [15] (the same concept as version vector – see Section 2.3.2) are used to minimize the number of updates exchanged between replicas. Each replica keeps a timestamp vector (TV), an N-element array of timestamps, that summarizes the state of the replica. Here, N is the number of replicas in the system, and timestamp is any number that increases monotonically – a logical clock [40], a wall clock, or a counter that increments at each update issuance. $TV[j]$ on replica j represents the time an update was last issued at j; therefore, $TV[j]$ is also called replica j's timestamp. Each update exchange among replicas is associated with its issuer's timestamp at the moment of issuance. When $TV[i] = v$ on replica j, j has received all the updates issued at i and with timestamps up to v.

Pull-based content-transfer update propagation is normally used in replicated file systems. To propagate updates, the system needs to decide which replica is the most up-to-date. Transferring the contents of the most up-to-date replica to others will bring all the replicas up to date. A simple way to do this is to use the *Thomas write rule* [58], which associates a timestamp with each replica. Each replica obtains a peer replica's timestamp, either periodically or on demand, and downloads the peer's contents only when its timestamp is older than the peer's. Given enough time for replicas to communicate with one another, all replicas will reach an eventual consistent state.

The Thomas write rule has one shortcoming: all conflicting updates are silently discarded except for the newest one. But systems designed for multi-user collaboration, such as Coda [37] and Ficus [27], need to know when updates conflict with each other to let users resolve conflicts or to resolve the conflicts automatically. To solve this problem, these systems associate a timestamp vector with each replica instead of a timestamp to detect conflicting updates.

Push-based update propagation systems are actually derivatives of pull-based counterparts. They differ only in the ways updates are distributed. Other issues can be addressed using the same strategies developed for pull-based systems.

2.3.2 Detecting and Resolving Conflicts

Version vector [49] and timestamp vector [15] are the same concept with different names [46]. In this work, data items are assumed to be files. An update to a file results in a new version of the file. Therefore, *version vector* is used instead of *timestamp vector*.

A version vector is an important data structure used by optimistic replication algorithms to keep track of updates to individual replicas and to reliably detect update conflicts. A version vector is an array of length equal to the number of replicas. An element of a version vector could be a simple counter. In the case of log-transfer systems, a version vector is attached to each update when it is issued to detect update conflicts; and in addition, a version vector is attached to each replica to keep track of the current state of the replica (described in Section 2.3.1). In content-transfer systems, it suffices to associate a version vector with each replica to detect version conflicts since each update results in a new version of a replica.

Each replica R_j independently maintains its own vector, and contains in position i the number of updates generated by replica R_i that R_j knows about. For example, assume there are three replicas and the version vectors are as follows:

$$\begin{array}{ll} \text{Replica 1} & V_1 = \{2,\ 0,\ 3\} \\ \text{Replica 2} & V_2 = \{2,\ 1,\ 3\} \\ \text{Replica 3} & V_3 = \{2,\ 1,\ 5\} \end{array}$$

Vector V_x is said to dominate V_y when V_x's elements are all pairwise greater than or equal to V_y's. In the example, V_3 dominates V_2, which dominates V_1. Therefore, replica 2 has more recent data than replica 1, and replica 3 is the globally most recent version. If replica 2's version vector had been $V_2 = \{2, 2, 3\}$, then V_2 would still dominate V_1, but V_3 would no longer dominate V_2. V_2 and V_3 are said to conflict in this case. A conflict resolver [39, 51] is needed to resolve the conflict, resulting in one of the version vectors dominating the other. If the conflict can not be resolved automatically, users will be notified to resolve the conflicts manually.

A *dynamic version vector* [49] reduces the space overhead by removing the zero value elements from the version vector, since the zeros do not affect the results of the comparison of the two version vectors. The modification makes the position of the dynamic version vector no longer imply the specific replica as it does in the basic version vector. A dynamic version vector, therefore, becomes a two-dimensional array with one dimension indicating the update counter and the other indicating the identification of replica. Because most of the updates are generated by a small number of the replicas, using a dynamic version vector can actually save storage space. In particular, when hosts are able to dynamically create or retire a replica of a file, dynamically maintaining the version vector is very important. After a replica is retired, a special garbage collection process is needed to remove the corresponding elements in the version vectors associated with this retired replica.

2.3.3 Reconciliation Topologies

When a replica selects another replica for reconciliation, it can use one of several reconciliation topologies. Although given enough time, the system can eventually reach eventual consistency by using pairwise communication, the reconciliation topology used affects update delivery latency, the degree of consistency between replicas, and the amount of network traffic caused by the reconciliation process.

Several reconciliation topologies have been studied [25]. The uniform random topology assigns every replica an equal probability of being selected as a reconciling partner. The distance-based topology always selects the nearest replica as a reconciliation partner. The advantage of this reconciliation topology is that it could reduce the traffic on the network. The selection can also be based on the cost of communication, perhaps measured in terms of latency or money. The adaptive ring topology is used in Ficus [27]. It predefines a ring of replicas. A replica always selects its next neighbour to reconcile. If the neighbour is not available at that time, it chooses the one following the neighbour, and so on, until it finds an available replica in the ring. The advantage is that it needs only $O(N)$ time to propagate the updates in the network.

There are some other reconciliation topologies which require the replicas to keep more information about replicas. For example, oldest-first topology uses a vector to record the times of reconciling with other replicas. Thus, a replica can always selects the replica with the oldest time as a reconciling partner by consulting the vector.

2.4 Previous Work

Data replication is an appealing technique to provide services on unreliable networks caused by poor connections, host failures, and network partitions. It has been studied for a long time in the areas of distributed file systems, distributed database systems, and recently peer-to-peer information storage and sharing systems. Since the focus of this study is using optimistic replication to support file sharing on mobile ad hoc networks, a brief overview of previous work on data replication is given below, focusing on how to replicate data and how to maintain replica consistency in the presence of file updates.

In the area of distributed file systems, optimistic replication is used to support mobile hosts. The Coda file system [37] is an optimistic replicated file system constructed on the client-server model. A mobile client is allowed to replicate files it needs during a disconnected period before disconnecting from the server. Such replications are managed by a cache manager, called Venus, which replicates critical files for the mobile host according to its file usage history. The file replicas replicated by Venus are used only by the mobile host itself, since mobile hosts are not allowed to communicate directly with each other. This restriction prevents effectively utilizing the file replicas in a mobile host's cache in the environment of a mobile ad hoc network, where hosts are supposed to assist each other to reduce the impact of network partitioning and slow wireless connections.

Ficus [27] adopts an optimistic peer-to-peer model with full replication. File updates are allowed to be performed on any file replica; and replica consistency is maintained by pairwise reconciliation processes, which is the same approach adopted in this study. Ficus requires users to replicate the same volumes[1] to be able to use an adaptive ring reconciliation topology. Volume replication could be a burden on resource-limited mobile hosts. To solve the problem, selective replication [50], which allows a user to only replicate files according to the user's interest, was added later in Ficus to support mobile hosts. Rumor [28], one of the descendants of Ficus, shares many similar features. It is a pure user-level file system, and supports selective replication to support mobile hosts. The reconciliation topology is multiple adaptive rings. A reconciliation process is done between two file replicas instead of between two volumes. This study shares many similarities with Ficus and Rumor for supporting file sharing. But Ficus and Rumor require every host to have knowledge about the replicated files on the other hosts to perform reconciliation processes. Such a requirement is difficult to realize on a mobile ad hoc network since hosts must dynamically replicate files they need to cope with the dynamic network topology. Although the same periodic pairwise reconciliation processes are used in this study, different reconciliation topologies (referred to as reconciliation policies in this study, see Section 3.3) are considered to alleviate problems of network partitioning.

XFS [3] is a serverless network file system. It utilizes the underlying well-connected high-speed network to group a cluster of clients. By distributing server functionality to every client in the cluster, it turns every client into a file server for some files. Although XFS is not a replicated file system, the idea of turning the clients into file servers and allowing clients to share their cache contents provides a way towards file system support for file sharing on an ad hoc network.

In distributed database systems, data replication is used to improve data availability and shorten response time. In general, a replica of a database is a full data collection of the database. Thus the replication strategy is not suitable for mobile ad hoc networks, where mobile hosts normally have limited storage space. To maintain data consistency among replicas, recently proposed methods are focusing on keeping updates in order using group communication primitives [35, 2] to main-

[1] A volume is smaller than the entire file system but larger than a directory.

tain strong consistency. Group communication primitives may not work well on frequently partitioning networks, and so such approaches are not suitable for mobile ad hoc networks either.

Using pairwise reconciliation processes to maintain replica consistency can also be found in replicated database systems, such as Bayou and Deno. In the Bayou [57, 48] system, data collections are replicated in full at a number of servers. Updates are allowed to be performed at a replica as tentative updates, and then propagated to the other replicas through pairwise reconciliation processes called anti-entropy sessions. Bayou maintains update orders by committing updates at a primary copy. Since this study is restricted to the file domain, updates are allowed to commit at any replica. Conflicts are resolved later. Deno [11] also fully replicates a data collection at a number of locations. It allocates a portion of currency to a replica. A reconciliation process for an update to a replica collects currencies through pairwise communication with other replicas. The process which collects more than 50% of the total currency commits the update. Those which do not gather enough currency are discarded. The committed updates are then propagated through pairwise communications. The currency mechanism used in Deno makes strong consistency guarantees (because the processes for the updates to the replicas of same data cannot gather more than 50% of the total currency at the same time) at the expense of committing fewer updates. Such an approach could also be used on mobile ad hoc networks if strong replica consistency is a requirement.

File sharing on a peer-to-peer network, as an overlay network on the Internet, has received much attention recently. Systems such as Gnutella [23] and FreeNet [14] are designed to explore the cooperations among hosts and enable file sharing among users on peer-to-peer networks. In Gnutella, a host does not have to share the file it gets from another host with the others unless it is explicitly required to do so. On the other hand, in FreeNet, files are cached automatically at the hosts who access them and are shared with other hosts. Thus, popular files will have more replicas than unpopular files. Neither Gnutella nor FreeNet deals with file updates. A new version of a file is simply treated as a new file. Therefore, no consistency maintaining methods are required.

There has been little research in data sharing and dissemination in mobile ad hoc networks. In [24], a broadcast method is used to disseminate popular files. Large files are broken into equal size file segments by applying the technique of Tornado coding [9]. A requesting host listens to the broadcasts and stores the file segments. Once the host gets all the file segments, it can reconstruct the file and become a source host to broadcast file segments, if necessary. The approach aims at speeding up file dissemination. But file updates are not considered.

In [34], Karumanchi *et al.* employed a variation of the quorum-based approach to support data sharing on a mobile ad hoc network. Predefined servers are divided into a number of quorums. Each quorum has at least one server in common. Updates are sent from a host to a quorum of servers, and a mobile host queries a data item from a quorum of servers to get the most up-to-date data. Such an approach is crucial when mobile hosts need accurate information. The cost is that some queries may fail. In this study, the assumption is that stale information is better than nothing. Thus a host is allowed to access a stale file. In addition, pairwise reconciliation processes are used to maintain weak consistency instead of the strong consistency enforced by a quorum solution.

In [29], Hara proposed three replica allocation methods to improve data accessibility on a mobile ad hoc network. Access frequencies to data items for each mobile host are assumed to be known in advance. Therefore, according to the data accessing frequencies and current ad hoc network topology, each host allocates a number of data replicas in its memory space. The allocation algorithm is periodically executed to cope with dynamically changing network topology.

The three allocation methods examined in [29] are: SAF (Static Access Frequency), DAFN (Dynamic Access Frequency and Neighborhood) and DCG (Dynamic Connectivity based Grouping). In SAF, each mobile host allocates data replicas in descending order of the access frequencies. Thus only the access frequency to each data items is taken into account. In DAFN, each mobile host allocates data replicas according to SAF first, then removes the duplicate replicas between neighbours. In DCG, an algorithm is executed to find the biconnected components in the network. In each component, the cache spaces of all hosts are treated as a big cache. Then, data replicas are allocated into the big cache according to the access frequencies.

This study has many similarities with [29]. Although both seek to improve data accessibility, this study considers file updates – something that has not been addressed in [29]. In addition, periodically allocating file replicas imposes a lot of load on the network. For this reason, this study focuses on on-demand allocation methods. Two methods, SAF and DAFN, are included for comparison purposes. Although The DCG method has been shown in [29] to have the best performance among the three methods, it has not been included in this study because it increases a significant amount of traffic on the network with a small amount of performance improvement. Furthermore, its dynamically computing biconnected components consume too much of the computing power of a mobile host to be practical in the real world. In addition to data accessibility, this study also investigates approaches to maintaining replica consistency in the presence of file updates, which has not been addressed in [29].

Chapter 3

System Description and Performance Issues

Given the special characteristics of ad hoc networks, it is very hard, if not impossible, to use traditional distributed file systems to support file sharing on ad hoc networks, because these file systems need powerful machines to provide server functionality and good network connectivity to keep *strong* consistency. With its resiliency to network failures and high data availability, optimistic replication has become one of the enabling technologies for mobile computing. In particular, optimistic replication enables a host which has a replica to perform any operations on the replica without knowledge of other replicas.

In this study, optimistic replication is used in combination with a peer-to-peer model to support file sharing between hosts on a mobile ad hoc network. In Section 3.1, Section 3.2, and Section 3.3, the system considered in this study and the proposed methods are described. The performance issues are addressed in Section 3.4.

3.1 System Model

A conference scenario was used as the basis for this study. A mobile ad hoc network is set up to support file sharing for attendees of a conference. Each mobile host holds a file as an original copy (the mobile host is the owner) which is to be shared with other hosts in the network. Each mobile host provides an amount of cache space, in which it is able to create replicas of files. Hosts move in the network area according to the mobility pattern described in Section 3.1.1. Since all the hosts are peers, a peer-to-peer optimistic replication model is employed which allows hosts to reconcile their file replicas with each other as needed in the face of file updates. The following sections explain how the system operates and how mobile hosts manage their replicas.

3.1.1 Mobility Pattern

The mobility pattern refers to how mobile hosts move in the network area. Given a value for the radio communication range of mobile hosts, the movement of mobile hosts is the main reason for the dynamic network partitioning and changing topology. Thus the mobility pattern affects the effectiveness of techniques for data replication and the maintenance of replica consistency [10].

Some random mobility patterns used in [52, 8, 33] for evaluating routing protocols may not suit this study. Since a conference situation is considered in this study, a group mobility pattern is used to simulate a conference situation. The network area is divided into four equal areas. Hosts are randomly divided into four groups. Each group of hosts moves in its own dedicated area according to a "random waypoint" movement model [8]. The speeds of hosts are chosen randomly from the range $(0, 1]$ as was done in [29]. After a period of time, hosts are randomly divided into four groups again. Each group is randomly reassigned to an area. Hosts then start to move to that area. Once they moved into their dedicated areas, they randomly move in that area until next the regroup time. The cycle continues until the end of the simulation.

3.1.2 Assumptions

A number of assumptions are made in developing the system for this investigation.

- The system is closed in the sense that the number of hosts in the network is fixed. Dynamic joining and leaving the network are not considered. Each host is the owner of a file, and does not create or delete files. Thus the number of original files is also fixed and equal to the number of hosts.

- Node failures are not considered. During the simulation time, all mobile hosts are fully functional. The radio communication ranges of the mobile hosts in the network are the same.

- The set of all mobile hosts in the network is denoted by $M = \{M_0, M_1, M_2, \ldots, M_m\}$, where m is the number of hosts in the network and M_j $(0 \leq j \leq m-1)$ is a host identifier.

- The set of all files is denoted by $F = \{F_0, F_1, F_2, \ldots, F_m\}$, where m is the number of files in the network and F_j $(0 \leq j \leq m-1)$ is a file's identifier (file F_j is owned by host M_j). Each file is assumed to be the same size. This is done for two reasons: 1) the size of a file affects only the number of file replicas a host can hold, which can be captured by changing the cache size of a host, 2) the elapsed time for transferring a file over the wireless link is not considered in this study (because this study does not address the issue of efficient utilizing wireless bandwidth).

- Each host has a cache space of N files, excluding the file owned by itself. Hosts are allowed to create file replicas into their caches. The access frequencies to files from each mobile host are known, and do not change.

3.1.3 File Accessing

When a mobile host issues an access request to a file (read or write operation), the request is successful if: 1) the mobile host is the owner of the file or has a replica of the original copy in the cache, or 2) the mobile host can contact the file owner or at least one mobile host which has a replica of the file in its cache. Thus, the host first checks whether it is the owner of the requested file or it has a replica of the requested file in its cache. If it does, the request succeeds. Then the host modifies its cache according to the replica management policy (see Section 3.2).

If the host does not have a replica of the requested file, it attempts to find the file and create a replica in its cache if it is necessary (according to the replica management policy used). It is possible for the host to ask the file owner for the requested file directly, but doing so has two disadvantages. First, the file owner may not be reachable at this time since the network may be partitioned at any time and the file owner may happen to be in another partition. In this case, the operation has to be aborted. This decreases the availability of the file service provided by the system. Second, the file owner may be beyond the transmission range of the host which is requesting the file. The high latency of wireless links will significantly decrease the performance of the system, and the multi-hop relayed communication will introduce more load on the network. Third, if the neighbour of the host happens to have a copy of that file, the host can lose the opportunity to get the file from the nearby host to reduce network load and shorten response time.

To overcome these disadvantages, a multicast protocol or a broadcast protocol could be used to locate the requested file. Because of their inherent location-transparency, these two kinds of protocols are very appealing for resource locating and retrieving. Ideally, if a file is associated with a multicast channel [26], and every host who has a replica of the file listens on the channel all the time, it will be easy to find the requested file by sending a message on the multicast channel. Unfortunately, multicast protocols for mobile ad hoc networks are built upon broadcast protocols [56], and in addition, require periodically maintaining multicast group membership. Therefore, using a multicast protocol to find files in a mobile ad hoc network may not save bandwidth. In particular, if the multicast group membership changes frequently, the multicast protocol will impose a lot of workload on the network in order to keep track of members in a multicast group. For this reason, a simple broadcast protocol is preferred to locate the requested file.

It is possible that multiple hosts have replicas of the requested file and will answer the broadcast request. This will cause redundant messages to be sent in the network. To avoid flooding the network, any host that is capable of answering the request prepares to do so but first initializes a randomized timer according to the round-trip delay. When the timer expires, it broadcasts its reply. The other hosts may cancel the timer upon receiving the reply. In this way, the response messages are suppressed, and the requesting host can always get a replica of the file from the nearest host. Once the requesting host locates the source location of the file replica, it pulls over the file data using a reliable unicast communication channel, and adjusts its cache contents. If the host cannot get the requested file, the access request fails.

3.2 File Replica Management

There are at least three advantages to replicating files in mobile hosts' caches and allowing hosts to perform operations on local replicas.

- Response time is decreased by avoiding possible multi-hop wireless communication.
- File accessibility is improved by using local file replicas in the presence of network partitioning.
- Less bandwidth is consumed since using a local replica avoids transferring the file from the file owner.

In any replica-based system, file replica management is an important issue, and system performance relies on it. In general, the more replicas that are spreading in the network, the higher file accessibility mobile hosts can get. A file replica management policy is used to decide which file replica should be created in a host's cache, and when. It is also used to decide which replica should be replaced (i.e., according to a certain replica replacement policy) if the cache is full and a new file replica needs to be created.

In this study, five replica management policies were considered: LRU-Only, FS-LRU, AF-LRU, SAF and DAFN [29]. The first three policies, LRU-Only, FS-LRU and AF-LRU, are on-demand approaches in the sense that a host creates a file replica in its cache when it successfully accesses the file after issuing the file access request. The last two policies, SAF and DAFN, are periodic synchronized approaches. In the two periodic approaches, file replicas are periodically allocated into hosts' caches at a certain replica allocation time for all the hosts in the network. During the time period between two consecutive replica allocation times, the replicas in a host's cache are relatively stable. The following describes the five policies in detail.

- LRU-Only (Least Recently Used replacement Only). LRU-Only is the simplest on-demand management policy. A file replica is allocated into a host's cache on-demand in the sense that a host creates a replica every time it successfully accesses a file which is not already in its cache. The newly created file replica is put on the top[1] of the cache, and the replicas in the cache are subject to LRU replacement (see Section 3.2.1) when the cache is full and the newly created replica must be allocated into the cache.

- FS-LRU (Fast-Spread combined with LRU replacement). LRU-Only may not fully utilize the cache space of a mobile host. First, a mobile host may not be able to successfully access all the files it requested because of network partitioning. Therefore, the cache space may not be filled up with replicas for a long time. Second, because of file updates, replica invalidation broadcasts cause stale replicas to be invalidated, resulting in empty spaces in the cache. Thus, cache space is not utilized effectively.

  To overcome the problem described above, the fast-spread method is used to improve utilization of the cache space of a mobile host. The fast-spread method allows a mobile host who is relaying a file in a multi-hop communication situation to create a replica of the file in its own cache space if the host has a certain amount of available cache space (20% of its cache size[2]).

[1] The replica at the top of the cache is the most recently used file replica. The replica at the bottom of the cache is the least recently used file replica.

[2] Although creating file replicas will not impose network load, the resources of a mobile host, such as computing power and battery power, are consumed. Preliminary experiments showed that 20% of the cache size is a reasonable value.

Thus the replicas of the file being delivered can be quickly spread into the network because multi-hop communications are common in ad hoc networks. The replicas created in this way are put at the bottom of the cache. When a host successfully accesses a file which is not already in its cache, a replica of the file is created and put on the top of the cache. Once the file replicas are allocated into the cache, they are subject to LRU replacement. One drawback of this approach is that a lot replicas of the same file may be created in the network.

- AF-LRU (Access Frequency combined with LRU replacement). In AF-LRU, a periodic frequency-based method is used to utilize the cache space of a mobile host effectively. Each mobile host checks the cache space in a specific period (allocation period). If there is a certain amount of cache space available (20%[3] of the cache capacity), some file replicas are created to fill up the cache space. The files are chosen according to the access frequencies which are local to a host and the content of the cache, i.e, the files with higher access frequencies for the host and currently not being in the host's cache are chosen to be replicated. If a chosen file is not accessible due to network partitions at the time, the cache space for that file remains unused. The newly created replicas are put at the bottom of the cache. If every host has the same allocation period and checks its cache at the same time, it is possible that every host tries to allocate file replicas at the same time to cause network congestion. Therefore, an unsynchronized approach is used. Every host has the same allocation period. But they may not check the cache at the same time so that the possibility of causing network congestion is avoided. When a mobile host successfully accesses a file which is not already in its cache, a replica of the file is created and put at the top of its cache, and replica replacement occurs at

[3]Preliminary experiments showed that 20% is a reasonable value. A large value will not fully utilize cache space, especially when the percentage of write requests in the workload is not high enough. A small value will impose too much load on the network, and the performance (file accessibility) does not improve that much.

the same time if the cache is full. Once the file replicas are in the cache, they are subject to LRU replacement.

- SAF (Static Access Frequency) [29]. In SAF, only access frequencies which are local to hosts are taken into account, regardless of the current topology of the network. Each mobile host allocates replicas in a specific period (allocation period). The replicas are chosen according to descending order of access frequencies. The number of replicas which can be allocated into the cache depends on the cache size and the current contents of the cache. File replicas whose updated flags are not set and whose access frequencies are not high enough will be removed from the cache. File replicas with updated flags set remain in the cache. At a time of replica allocation, a mobile host may not be able to get the required file replica. In this case, the cache space for that replica is not used. The file replica is created when an access request to that file replica succeeds or when the mobile host can get it from another host at another allocation time. A read request will not cause a replica to be created. But in order to support file update, a successful write request will cause a replica of the file to be created if one is not already in the cache. At that time, if the cache is full, the file replica whose access frequency is the lowest and whose updated flag is not set is removed to make room. Therefore, once the cache is full of file replicas, the cache is relatively stable since read operations do not cause the cache contents to change.

- DAFN (Dynamic Access Frequency and Neighborhood) [29]. In DAFN, file replicas are periodically allocated. The network topology and the access frequencies (which are local to hosts) are taken into account. At an allocation time, file replicas are allocated according to the SAF policy first, then file replica duplication[4] between two neighbours is removed. In each set of mobile hosts which are connected together, a breadth-first search starting with the

[4]File replica duplication is a situation where two hosts have replicas of the same files, or one host has a replica of the file whose owner is the other host. A mobile host should not have file replica duplication in its own cache.

lowest suffix (i) of host identifier (M_i) is used to remove replica duplication between two neighbours. When there is file replica duplication between two neighbouring mobile hosts, and if one host is the owner of the file, the other host changes the replica to another replica. If both hosts hold replicas of the same file, the one whose access frequency to the file is lower changes the replica to another replica.

Figure 3.1 shows an example of replica allocation for the four hosts according to the DAFN policy. The access frequency of each mobile host to each file is shown in Table 3.1. After allocating replicas according to the SAF policy, replica duplication between two neighbours is removed. A light grey box denotes a file owned by the host, A dark grey box denotes a replica allocated for removing replica duplication. The changes for removing replica duplication are:

Table 3.1: Access Frequencies to files

Files	Mobile host			
	M_1	M_2	M_3	M_4
F_1	0.27	0.16	0.19	0.13
F_2	0.18	0.30	0.12	0.09
F_3	0.04	0.12	0.15	0.21
F_4	0.06	0.9	0.32	0.14
F_5	0.14	0.08	0.2	0.08
F_6	0.3	0.10	0.06	0.11
F_7	0.12	0.7	0.07	0.06
...	...	...	...	...

M_1-M_2: $F_1 \rightarrow F_6$ (M_2), $F_2 \rightarrow F_7$ (M_1)

M_1-M_3: $F_1 \rightarrow F_2$ (M_3)

M_2-M_3: $F_2 \rightarrow F_1$ (M_3), $F_3 \rightarrow F_7$ (M_2)

M_3-M_4: $F_3 \rightarrow F_6$ (M_4), $F_4 \rightarrow F_2$ (M_3), $F_1 \rightarrow F_5$ (M_4)

Where, M_i-M_j denotes that the replica duplication is eliminated between two mobile hosts M_i and M_j, and $F_k \rightarrow F_l$ (M_i) denotes that the replica allocation is changed from F_k to F_l at host M_i. While replicas of five different files are allocated among hosts according to the SAF policy, replicas of seven different

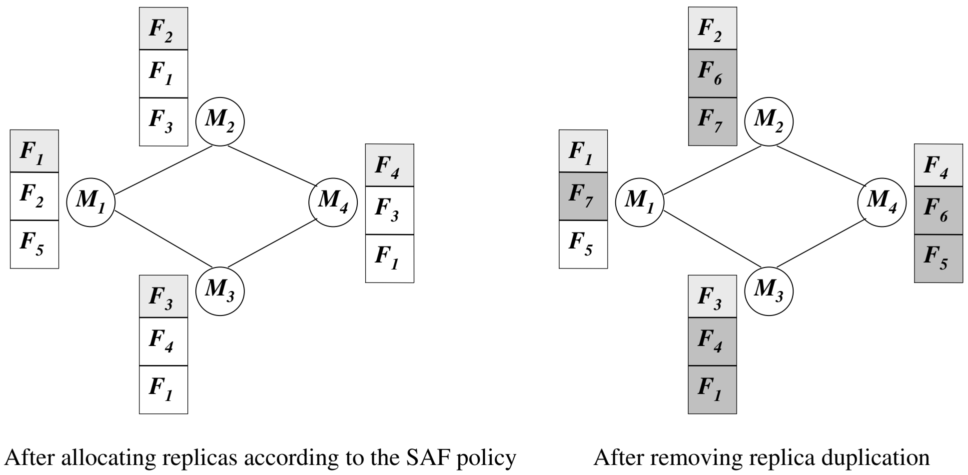

Figure 3.1: An example of replica allocation according to the DAFN policy

files are allocated after removing replica duplication. Because DAFN deals with two hosts at a time, the replica duplication cannot be removed completely. Nevertheless, DAFN should provide better performance than SAF because of the effort of removing replica duplication.

At an allocation time, the required file replica may not be available. In this case, the cache space is unchanged. The required file replica is created a) when an access request to the file succeeds, or b) at another replica allocation time if it is still needed and becomes available at that time. The write operations have the same effect in DAFN as in SAF.

3.2.1 Replica Replacement Policies

A replica replacement policy is used to decide which file replica should be removed from the cache when the cache is full of replicas and a new file replica needs to be created. Under on-demand replica management policies, file replicas are created whenever hosts successfully access files which are not already in their caches. Thus, a host's cache can be quickly filled up, particularly when cache size is small and the amount of write operations in the workload is also small. Different replacement policies will cause different replicas to reside in the cache, and hence, result in different performance.

Unlike a traditional cache replacement policy [60, 21], which is used to maximize the benefit to the host that uses it, a replica replacement policy is used not only to benefit the host that uses it, but also to assist the other hosts in the network. In this study, the performance impacts of four replica replacement policies are examined. The following describes the four policies in detail:

- LRU (Least Recently Used). The order of the replicas is maintained by putting the most recently used replica onto the top of the cache, and keeping the least recently used replica at the bottom. If a file access request succeeds because of a cache hit, the file replica is brought onto the top of the cache; if the request succeeds and causes a replica to be created, the newly created file replica is put onto the top of the cache. In the case that the cache is full, the least recently used file replica (which is at the bottom of cache) whose *updated* flag is not set is removed from the cache to make room for the newly arrived file replica.

- LFU (Least Frequently Used). A reference counter is maintained for each file replica in the cache. If a file access request succeeds because of a cache hit, the counter of that file replica is increased by 1; if the request succeeds and causes a file replica to be created, the replica is put into the cache and the counter for that file replica is set to 1. If the cache is full, the least frequently used file replica (i.e., the one with the lowest counter) whose *updated* flag is not set is removed to make room for the newly arrived file replica.

- LAF (Lowest Access Frequency). In LAF, the file replicas in the cache are managed according to the access frequencies to the file replicas (which are assumed to be known in advance). When the cache is full and space is needed for a newly arrived file replica, the file replica which has the lowest access frequency among the file replicas and whose updated flags are not set in the cache is removed.

- DWF (Dynamic Write Frequency). Since frequently updated files are not good candidates to replicate in a mobile host's cache, DWF replaces the file replica that is most frequently updated. Each mobile host maintains a list of counters for all the files in the system. A counter indicates the number of times a file has been updated. When a mobile host issues a write to a file replica in its cache, the counter of the file corresponding to the file replica is increased by 1. After the mobile host modifies the file replica, it broadcasts a invalidation message[5] to inform other hosts in the system to invalidate the stale file replicas of that file. The mobile hosts that receive the invalidation broadcast increase the counter of the file and invalidate the file replica if it is in their caches. If a mobile host does not receive the invalidation broadcast because of network partitioning, the counter it maintains is unchanged. Thus, the counters of a file maintained by different hosts may have different values. When a host needs to remove a file replica to make the room for a newly arrived file replica, it chooses to remove the file replica with the highest corresponding counter value. A tie is broken by choosing the one that enters the cache earliest.

3.3 Maintaining Replica Consistency

In the presence of file updates, some level of consistency must be maintained among all the replicas. Since any operation can be performed on a local file replica, there is a risk of permitting concurrent updates and serving stale files to applications. In many scenarios, however, the risk is acceptable for three reasons. First, most files have a single writer, particularly over a short time period. Thus allowing writes to any file replica rarely results in concurrent updates [37]. Second, a large class of applications (e.g., library database systems) can work well with slightly aged information, and immediate propagation of new updates to all replicas is not vital. Third, for many applications, update conflicts can be resolved without data loss either automatically [39, 51] or manually. The directory file is an important example. Independent file creations can be applied to two replicas of a directory file without causing problems,

[5] The details of invalidation broadcast are given in Section 3.3

because the different directory replicas can be easily merged into a single directory.

Update conflicts among replicas are detected by comparing *version vectors* (see Section 2.3.2) associated with each file replica. Eventual consistency is provided by periodically initiating *reconciliation processes*. Periodically performing reconciliation is necessary for an isolated host to get file updates once it reconnects to the network.

Although every host has the same reconciliation period, different hosts may initiate reconciliation processes at different times, i.e., hosts start their first reconciliation process at different times, then adopt the same reconciliation period. Each host performs a reconciliation process for one of its file replicas during its reconciliation period. This means that the host needs to perform the same number of reconciliation processes as the number of file replicas in its cache. For example, if a host keeps five replicas in its cache, it needs to perform five reconciliations to bring all the replicas up to date.

At a reconciliation time, a mobile host needs to choose another mobile host as its reconciliation partner. Which one a mobile host will choose is controlled by the reconciliation policy (see Section 2.3.3). Using different reconciliation policies will result in different levels of consistency among file replicas. In this study, two reconciliation policies are investigated.

- CH (Closest Host). A host chooses its available closest host to reconcile the specific file replica. At a reconciliation time, a host broadcasts the file identifier and its version vector associated with the file replica to find the closest host who has a replica of the same file. When a host receives the message, it checks its own cache to look for the corresponding file replica and its version vector. Then it does one of the following:

  – If the receiving host has the file replica (or it is the owner of the file) and its version vector dominates (or is dominated by) the received version vector, it will reply to the message. Once the two hosts know who has the newer version, the contents of the newer version will be transferred through a unicast channel to bring the old version up to date and at the same time the version vectors are updated.

  – If the receiving host's version vector conflicts with the received version vector, the two hosts need to negotiate with each other to resolve the conflict, either through an automatic resolver or manually. It is assumed that the resolved file version along with the resolved version vector will reside in both hosts. The resolved version vector contains the same number of elements which are the pairwise maximum of both version vectors.

  – If the receiving host neither has the corresponding file replica nor is the owner of the file, it just ignores the message so that the appropriate host can reply.

  One optimization used in this study is that a host records the hosts with which it reconciled last time, and so the following reconciliation will not consider them as reconciliation candidates. This optimization will speed up the propagation of updates in the network when the file update rate is not high.

- FO (File Owner). A host chooses the host who is the file owner to reconcile the specific file replica. At reconciliation time, a host contacts the file owner directly by sending the owner the version vector of the file replica using a unicast channel. Upon receiving the version vector, the file owner compares the received version vector with its own version vector and reconciles the file with the sending host in the same way as they do in CH above.

  It is possible that the file owner cannot be reached because of network partitions. If so, the host simply gives up the reconciliation on this replica and goes on with the next file replica.

Initial experiments showed that relying only on periodic reconciliation processes is not fast enough to propagate updates, which causes many older version files to exist in the system. To speed up the propagation of updates, one possible way is to broadcast the updated file in the network after a host finishes its update to a file, as was done in [27]. In a mobile ad hoc network, however, the bandwidth of wireless communication is limited. Therefore, a *broadcast invalidation* method is used to invalidate stale file replicas. When a host finishes updating a file replica, it broadcasts an invalidation message along with the file identifier and its version vector. Upon receiving the message, the host checks its cache and performs one of the following:

- If the host has a replica in its cache and its version vector is dominated by the received version vector, the host removes the file replica from its cache. If the two version vectors conflict with each other, the host will contact the sending host to resolve the conflict.

- If the host is the owner of the file, and its version vector is dominated by the received version vector, it will contact the sending host to get the newer version of the file immediately because the file owner should get the modified file as soon as possible. If they conflict with each other, the file owner will contact the sending host to resolve the conflict immediately for the same reason.

3.4 Performance Issues

3.4.1 Performance Metrics

In general, the objective is to provide high file accessibility and short response time, while using less bandwidth. In the presence of file updates, updates should be propagated quickly and efficiently to keep file replicas consistent with each other. To evaluate the performance of various policies, the following metrics are used in this study.

- **File Accessibility** – File accessibility is defined as the percentage of file access requests (including read requests and write requests) in the whole workload that are successful. A file access request is successful when the host that initiated a file request is able to accesses the file or one of its replicas. In a traditional LAN environment, all hosts are strongly connected with each other, and so file accessibility will be 100% in the absence of host failure. In a mobile ad hoc network, the requested file may be in another partition since the network can get partitioned at any time, which causes the file inaccessible. Thus file accessibility is degraded. High file accessibility is a primary goal of any data replication system.

- **Fresh Access** – File accessibility measures overall data availability, but says nothing about the quality of the data accessed. Since optimistic replication is used to support file sharing among the hosts in the network, file replicas are not consistent all the time. Therefore, a mobile host may access stale file replicas, and stale write accesses will eventually create conflicts among file replicas.

  Fresh access is a measure of quality of service. It is defined as the percentage of successful access requests (including read requests and write requests) to the most up-to-date file replicas in the whole workload. A successful access request to a particular file replica is fresh if the version vector of the file replica is not dominated by any of the other version vectors of the file replicas of the same file.

- **Average Number of Hops per Access** – This measure reflects the response time for a host accessing a file. It is defined as the ratio of the total number of hops required to access files to the total number of access requests in the workload. The total number of hops includes both successful access requests and unsuccessful (or failed) access requests since an unsuccessful access request will cause the request issuing host to block. In this study, the hop count for a failed access request is counted as the diameter of the network measured in hops, plus 1 hop. The diameter of the network is a function of radio communication range, and can be expressed as $D = L/R$ where D is the network diameter in hops, L is the maximum length of the two points in the network area, and R is the radio communication range of the wireless interfaces of the mobile hosts.

In a mobile ad hoc network, multi-hop communications are expected to be common. Since a wireless link usually has lower bandwidth than a wired link, it takes longer to transfer a file over wireless links than over wired links. In addition, the more hops a route contains, the more likely the route will be broken during file transferring periods. In general, a lower average number of hops per access means that a user spends less time waiting for arriving files.

- **Total Hops** – Total Hops is defined as the total hop count of file transmission including allocating and accessing file replicas. This measure reflects the total network load imposed on the network by a file replica management policy. Although the primary goal of file replication is to improve file accessibility, imposing little network load is also an important requirement in a mobile ad hoc network environment since a wireless link has lower bandwidth than a wired link. Therefore, a good file replica management policy should provide high file accessibility and impose little load on the network.

- **Conflict Ratio** – This is a popular measure for replicated systems [38, 44]. It is a global measure defined as the ratio of the number of conflicting write accesses to the total number of write access requests in the workload. The lack of restrictions on update patterns in optimistic systems leads to conflicts, in which two or more replicas are updated based on a common ancestral version (as opposed to having one update based on the results on the other). The advantage of this metric lies in its simplicity: it is easy to understand. A system with few conflicts is desirable because it requires little effort to resolve the conflicts and usually adheres to single-copy semantics.

  One difficulty associated with conflict ratio is when to count a conflict. In a real system, a conflict can be counted only at reconciliation time. Thus the frequency of reconciliation affects conflict ratio. In the extreme, if reconciliation never occurs, the conflict ratio will be zero. A simulation environment provides more control, and a conflict can be counted at every write access.

For example, a stale write will cause a conflict at reconciliation time. In this study, once a write access to one of the replicas of a file succeeds, the version vector of the replica is compared with the version vectors of the other replicas of the file. If they are conflicting or if the version vector is dominated by the other version vectors (i.e., the write access is a stale write access), the conflict count is increased by one.

- **Reconciliation Hops** – This measure reflects the network load imposed by all reconciliation processes for maintaining replica consistency. Different reconciliation policies will result in different total number of reconciliation hops. It is defined as the total number of hops used for data transmissions for reconciling file replicas during the whole simulation time. Although the purpose of reconciliation is to keep consistency among file replicas, imposing little network load is also an important requirement on a mobile ad hoc network because of the limited bandwidth of the wireless links.

3.4.2 Factors Affecting Performance

Given the complex characteristics of mobile ad hoc networks, many factors affect the performance of a policy used to support file sharing on the network. In this study, the following factors are examined:

- **Workload** – The performance of any replication system is significantly impacted by the workload presented to it. The characteristics of workloads, such as the temporal locality of accesses and the distribution of reads and writes, have a significant impact on performance. Unfortunately, no real workloads have been collected from an ad hoc network, and so synthetic workloads were used to provide the input to the simulator in this study. This has the benefit of greater experimental control. In particular, the percentage of write accesses can be varied in the workloads to study the performance impact of file updates.

- **Replica Management Policy** – The replica management policy (see Section 3.2) determines when to create a file replica and when to replace a file replica. Thus, different replica management policies will result in different file replicas residing in a host's cache. An ideal policy would spread file replicas evenly in the network to improve file accessibility, and an improved cache hit ratio would provide shorter response times.

- **Replica Replacement Policy** – As a part of a replica management policy, a replica replacement policy determines which replica will be removed when the cache is full. Therefore, a replica replacement policy affects the distribution of file replicas in the network. Four replica replacement policies were considered in this study, and their performance impacts were investigated.

- **Reconciliation Policy** – A reconciliation policy is used to decide which host will be chosen as a reconciliation partner. The policy chosen directly affects the way updates propagate, and therefore has a significant impact on how file replicas are kept consistent. In addition, different reconciliation policies will impose different amounts of network load on the mobile ad hoc network where the bandwidth of wireless links is limited.

- **Cache Size** – The cache size is always an important factor in any storage system. With a large cache, a mobile host can store more files locally, and thus can work more independently in a mobile ad hoc network. But a large cache means more file are cached and means that there are more replicas in the network, which creates difficulties in maintaining consistency among the replicas. Nevertheless, a large cache is likely more desirable than a small cache.

- **Radio Range** – Mobile hosts communicate with each other through wireless links. The radio range of the wireless interfaces of mobile hosts affects the network connectivity. The smaller the radio range, the more frequently the mobile ad hoc network gets partitioned. For example, if the radio range can cover the whole network area, there will never be partitions in the network and file accessibility will theoretically reach 100%.

- **Allocation Period** – The allocation period refers to how often the cache is checked and replicas are allocated as necessary in the three periodic replica management policies, AF-LRU, SAF and DAFN. In AF-LRU, replicas are allocated only when there is enough empty space in the cache (20% of cache space). In SAF and DAF, the cache contents are relatively stable during the period of two consecutive allocation times. Therefore, the allocation period should have little effect on the performance of the three policies.

- **Reconciliation Period** – The reconciliation period refers to how often a mobile host will initiate a reconciliation process. A small reconciliation period will cause a host to initiate more reconciliation processes, thus making updates propagate faster. But this will impose some unnecessary amount of load on the mobile ad hoc network since some reconciliation processes will not get new information if the update rate is low. If the reconciliation period is too large, too many stale accesses may result.

Chapter 4

Simulation Environment

In order to evaluate the performance of various replica management policies and reconciliation policies, a simulation study was conducted. In this chapter, the details of the simulation model and how the synthetic workload was generated for this study are described. At the end of the chapter, the simulator and its configuration for this study are discussed.

4.1 Synthetic Workload

The performance of any system depends heavily on the workload presented to it. Although the design principle for any system is to achieve the best performance under all possible workloads, this goal is rarely achieved in reality simply because there are unlimited variations in workload. In practice, a system usually ends up being optimized for certain kinds of workload.

Because no workload traces were available from a real mobile ad hoc network, it was decided to develop a workload generator to create synthetic workloads for this study. Although a synthetic workload cannot represent a real workload exactly, it offers several experimental benefits since it is easy to vary the characteristics presented in the synthetic workload in a controlled way. There are existing synthetic workload generators in the literature [17, 5], but all server-oriented.

This section explains how the synthetic workload was generated and its important characteristics.

4.1.1 Determining a Request

The synthetic workload required for this study is simpler than that typically used in studies of distributed systems, since file creation and file deletion are not considered. The information provided in a request is summarized in Table 4.1.

Table 4.1: Request description

Name	Description
Host ID	The host who issued a file access request
Time	The time the request is issued
Operation	Read or write access
File ID	The file to which the host issued a request
File Owner ID	The host who is the owner of the file

The following describes how each component of a synthetic request is determined:

- **Host ID** – The workload generator generates a stream of requests for one host at a time. To generate all the requests, the generator is run iteratively a total of N times, where N is the number of the hosts in the system. The ith run (i is in $[0, N-1]$) is for generating the workload for the mobile host M_i, and so i is recorded as the host ID. The final workload is formed by combining all the requests for all hosts and sorting them by time.

- **Time** – The time of a request is determined by the inter-arrival time between two consecutive access requests issued by a host. This is modelled as an exponentially distributed random number. The mean inter-arrival time determines the number of requests in a time period and thus the total number of requests in the workload for a given simulation time. Since replica consistency can be adjusted by varying the reconciliation period (see Section 3.3), the characteristics of the inter-arrival time distribution have little effect in this study.

- **Operation** – The type of operation requested is determined by the pre-defined ratio of read operations to write operations. A uniformly distributed random number between 0 and 1 is generated and compared to the ratio to decide whether this operation should be a read or a write.

- **File ID** – Different hosts are usually interested in different sets of files. The access frequencies of each mobile host to all the files in the system are computed according to a certain rule (see Section 4.1.2). The File ID is determined by comparing a uniformly distributed random number to these file access frequencies.

- **File Owner ID** – In this study, each host is allowed to have exactly one original file to be shared with the other hosts in the network. Therefore, the file owner ID is the same as the file ID.

The number of requests in the final workload is determined by the length of the simulation.

4.1.2 Important Characteristics of a Workload

Three important characteristics are considered when generating the workload.

- **The concentration of file accesses**: The concentration characteristic of a workload refers to the non-uniform file accessing behaviour [4, 59]. It is an important factor that affects file accessibility. For example, if all the hosts are interested in only a few files, hosts can always replicate files locally, and the expected file accessibility will be high.

  To reflect the concentration characteristic, the file access frequencies for each host are determined as was done in [29]. The access frequency of a mobile host to file i is f_i, determined as a positive value based on the normal distribution with a mean of $0.5(1 + 0.01i)$ and a standard deviation of 0.01.

- **The set of files of interest**: Mobile hosts usually have different sets of files of interest (which will be referred to as a host's *interest set*). Although the concentration characteristic represents a situation where some files are more popular than others, a particular mobile host may have its own interest set. The difference among the interest sets of the mobile hosts is determined by the standard deviation of the normal distribution used to compute the access frequencies for the mobile hosts. The larger the standard deviation, the larger the difference among the interest sets. In [29], it was shown that a large standard deviation makes the performance difference between replica management policies too small. Thus, in this study, the standard deviation is chosen to be 0.01.

- **Temporal locality**: Temporal locality means that files being accessed are likely to be accessed again in a short time. This is a well known characteristic of client workloads. Following the approach used in [1], the temporal locality can be expressed as shown in Table 4.2.

Table 4.2: Temporal locality parameters used by workload generator

Probability	File chosen from
0.3	Files referenced in previous 1500 time units
0.4	Files referenced in previous 3000 time units
0.6	Files not referenced in previous 3000 time units

The values of the probabilities in Table 4.2 determine how strong the temporal locality is in the workload generated by the workload generator. When choosing a file for an access request, a random number in (0, 1) is generated. With probability 0.3, the file requested is selected from the set of files that were accessed in the most recent 1500 simulation time units; With probability 0.4, the file requested is selected from the set of files that were accessed in the most recent 3000 simulation time units; With probability 0.6, the file requested is selected from the set of files that were not accessed in the most recent 3000 simulation time units. The values in Table 4.2 are determined by the analysis of the other outputs from the generator. For example, on average, there is 30% chance for a host to choose a file from 16%[1] of the total files in workload. Preliminary experiments showed the values chosen in the table have a moderate effect on the temporal locality in the workload.

- **Percentage of writes**: This parameter reflects how many write operations there are in the workload. One of the goals of this study is to investigate the performance impact of file updates under different file replica management policies and different reconciliation policies. Different workloads are generated by varying the percentage of writes and keeping other parameters unchanged.

4.2 Simulation Model

The simulation model is developed according to the mobile ad hoc network system described in Chapter 3. A brief description of the simulation model is given below.

Four components are modelled:

- **Mobile Host** – A mobile host is modelled by three parameters: radio range, speed, and moving area. A group mobility pattern is used to simulate a conference situation, which is described in Section 3.1.1. Each mobile host has a local cache.

- **Network** – Mobile hosts communicate with each other through wireless connections (single-hop or multi-hop) without any interference. Since this study does not address problems related to bandwidth, a file transfer from one host to another takes no time. The connectivity of the network depends on the positions of hosts in the network area and the radio communication range of the mobile hosts.

[1] On average, it was shown in the workload generator that there are 7 different files having been accessed in the last 1500 simulation time units. The total number of different files in the workload is 40. Therefore, there is around 16% of total different files accessed in the last 1500 simulation time units.

- **File** – A file is modelled as a record containing information about the file's identity, the owner's identity, update information, and a version vector. A file is fixed in size and all files are the same size.

- **Local Cache** – Local cache is modelled as a collection of entries (files) and a few bookkeeping variables. For the purpose of replacement, the entries may be sorted according to recency of reference, frequency of reference, static access frequency, or write frequency. The cache size is fixed and all hosts have the same cache size during a simulation experiment. Since all files have the same file size, the cache size is expressed as the number of files it can hold.

The interactions between the components are modelled by six processes:

- **File Access** – The file access activities of the mobile hosts are driven by file access requests. This process reads one request at a time from the workload used by the simulator, and sets up the time for the host to perform read or write access to the file according to the request. Upon finishing the access, the process reads the next request. Depending on the request and the network conditions at the time, a file replica may be created and a replica replacement may occur.

- **Change in Network Connectivity** – This process calculates the network connectivity matrix, i.e., which mobile hosts a host can connect to at the time. It also sets the next time to recalculate the connectivity matrix when a link between any two mobile hosts is broken.

- **Replica Management** – This process decides which file replica should be created and when, according to the policy used. Five policies are modelled and these are described in Section 3.2.1.

- **Replica Replacement** – Replica replacement occurs when a new replica needs to be created but the cache is full. Four replica replacement policies are modelled and these are described in Section 3.2.1.

- **Reconciliation** – This process is invoked to keep consistency among the file replicas. It occurs periodically depending on the reconciliation period set up at the beginning of the simulation. The reconciliation process determines which host should be chosen as a reconciliation partner based on the reconciliation policy used. Two reconciliation policies are modelled and these are described in Section 3.3.

- **Invalidation Broadcast** – In order to speed up update propagation, an invalidation broadcast is used. It occurs 1) when a host successfully performs a write access; 2) when two hosts resolve two conflicting file replicas during a reconciliation process (see Section 3.3).

4.3 The Simulator

The simulator was implemented as a discrete event-driven simulator written in C++. The object-oriented design of C++ makes it easy to test a single object and to add more functions to an object as the simulator is developed.

Validation of the simulator was done in two parts. First, results from the simulator were compared against those shown in [29]. This was done by using the same parameter values as used in [29] and a workload with 0% writes. Both results were very similar. In the second part, by using the small "workload"– truncated version of workload with 10% writes used in this study, detailed output from the simulator was checked for the correctness as each request was processed.

The simulation time was set to be 100,000 time units. This is long enough for the simulator to process roughly 100,000 requests. Since results obtained for 100,000 time units were very similar to those obtained for 50,000 time units, this run length was deemed sufficient.

4.3.1 Parameter Settings

The simulator is capable of simulating different situations by changing parameters. Table 4.3 shows the values of fixed parameters for the general configuration of the simulator. All experiments for this study were conducted under this configuration.

Table 4.3: Parameter values for the simulation experiments

Parameters	Values
Network area	50 X 50
Speed range for mobile hosts	$(0, 1]$
Number of hosts	40
Number of original files	40
Simulation time	$100,000$ time units

Values for fixed parameters were chosen to be the same as those used in [29] so that some results from the simulation experiments could be used to verify the correctness of the implementation of the simulator. Preliminary experiments showed that these parameter settings were reasonable choices.

Parameters, such as radio range, cache size, relocation period, and reconciliation period, were varied in different simulation experiments to examine their impact on performance. In the experiments, the results reported were obtained by averaging results from three runs of the simulator with different random seeds. Although different random seeds caused different results, the rank order of policies and the relative differences among the policies examined were very similar.

4.4 Road Map for Experiments

To evaluate the performance of different policies, three sets of experiments were conducted. Experimental results are presented in the following three chapters. Each chapter focuses on one policy area. Chapter 5 addresses replica management. Five policies, AF-LRU, FS-LRU, LRU-Only, SAF, and DAFN, are examined with CH as the reconciliation policy. Chapter 6 addresses replica replacement. Four policies, LRU, LFU, LAF, and DWF, are examined with CH as the reconciliation policy. Chapter 7 addresses replica reconciliation. Two policies, CH and FO, are investigated with AF-LRU as the replica management policy.

Chapter 5

Experiments with Replica Management Policies

When a mobile ad hoc network gets partitioned, the hosts in a particular partition must rely on the file replicas residing in the hosts' caches in the partition to keep working. Therefore, to effectively support file sharing on a mobile ad hoc network by using optimistic replication, it is crucial for a host to make decisions on when to create a file replica, which file to replicate, and which file to replace when the cache is full. This is referred to as replica management. In the presence of file updates, the replica management policy should also provide a reasonable file access rate to the up-to-date file replicas. In addition, to reduce the response time, file replicas should be kept where they are most likely to be used.

This chapter presents the results of the performance study of different replica management policies. The experimental design is described and the experimental results are discussed.

5.1 Experimental Design

5.1.1 Policies Investigated

Five replica management policies (see Section 3.2) were examined:

- LRU-Only (Least Recently Used replacement Only) – The LRU-Only policy imposes less load onto the network than the other policies because a host replicates only files it has already accessed. However, the cache space may not be utilized effectively when the file update rate is high.

- FS-LRU (Fast-Spread combined with LRU replacement) – The FS-LRU policy is the fastest way to spread file replicas into the network. The drawback is that it may generate too many replicas of one file, and so may not utilize cache space effectively.

- AF-LRU (Access Frequency combined with LRU replacement) – In AF-LRU, files are replicated when the host accesses a file and when there is enough empty space in the cache. The cache space is utilized more effectively in AF-LRU but more load is imposed on the network than in LRU-Only and FS-LRU.

- SAF (Static Access Frequency) and DAFN (Dynamic Access Frequency and Neighborhood) – In this two policies, file replicas are allocated periodically. During the time between two consecutive allocations, the contents of the caches are relatively stable. This may not suit the dynamic nature of a mobile ad hoc network.

5.1.2 Performance Metrics and Factors

Four performance metrics are used to evaluate the performance of the different replica management policies: file accessibility, fresh access, average hops per access and total hops. These are described in Section 3.4.1.

Four important factors, cache size, write percentage, radio range and allocation period, are varied over wide ranges to examine their impact on the performance of the replica management policies.

5.1.3 Parameter Settings

The parameters used in the simulation experiments are shown in Table 5.1. To maintain replica consistency, the CH (Closest Host) reconciliation policy is used, in which a host selects the closest available hosts as reconciliation partners (see Section 3.3). CH was chosen since it provides better performance than FO (File Owner) does (as shown in Section 7.2), in which a host selects file owners as reconciliation partners. Since a smaller value of reconciliation period results in a higher level of replica consistency, the reconciliation period for CH is set to a relatively small value

Table 5.1: Factors and levels used in the simulation experiments

Factors	Levels
Replica management policy	LRU-Only, FS-LRU, AF-LRU, SAF, DAFN
Reconciliation policy	CH
Reconciliation period (time units)	20
Cache size	2 to 38 files, in steps of 2 files
Write percentage in workload	2%, 4%, 6%, 8%, 10%, 15%, 20%, 25%, 30%
Radio range (distance units)	1, 3, 5, 7, 9, 11, 13, 15, 17, 19
Allocation period (time units)	$5 * 2^n$ where n is from 1 to 13

– 20 time units, which means that a host will initiate reconciliation processes to reconcile its file replicas with the partners frequently. A large range of cache sizes is used to show how cache size affects the performance of replica management. In order to investigate the impact of file update on the performance of replica management, the percentage of write operations in the workload is varied from 0% to 30%. A write percentage of 30% in the workload is high enough for a stress test in this study. Radio range is an important parameter in a mobile ad hoc network, since it affects the frequency of network partitioning which is the main cause of low file accessibility. The preliminary experiments showed that setting radio range to be 19 distance units in a flat 50x50 network area (see Section 4.3.1) causes few network partitions throughout the simulation time. Thus the largest value of radio range examined was 19 distance units. The range of values for allocation period used by AF-LRU, SAF, and DAFN was chosen to be large enough to show the impact of the allocation period on the performance of file replica management.

Experiments were conducted by varying the value of one parameter at a time while keeping the others fixed at specific values. Table 5.2 shows the values of parameters when they were fixed. The values of parameters were chosen to make the difference among the performance of replica management policies easy to see.

Table 5.2: Fixed values for parameters

Parameter	Value
Cache size	10 files
Radio range	8 distance units
Write percentage in workload	10%
Allocation period	512 time units

5.2 Experimental Results

Four sets of experiments were conducted by varying different parameters to show how these factors affect the performance of replica management policies. The results are presented and discussed in the following four sections.

5.2.1 File Accessibility

File accessibility is an important goal of any data replication system and relies heavily on the effectiveness of file replica management. High file accessibility is always desirable.

Figure 5.1 shows file accessibility as a function of cache size for the five replica management policies considered. Not surprisingly, increasing cache size benefits file accessibility for all five policies, and because of network partitions, none of the five policies reaches 100% file accessibility only by increasing cache size. In general, when cache size is small, increasing cache size has a dramatic impact on file accessibility for all policies. But when cache size reaches a certain value, further increases have little effect on file accessibility. This suggests that there is a "best" cache size for each replica management policy, and different policies have different best cache sizes. For example, the best size for LRU-Only is around 12 files, but SAF requires around 34 files.

SAF gives the worst performance among the five policies considered when the cache size is smaller than 28 files. This is because replicas are allocated in the cache based only on the access frequencies to data items. If hosts have similar access

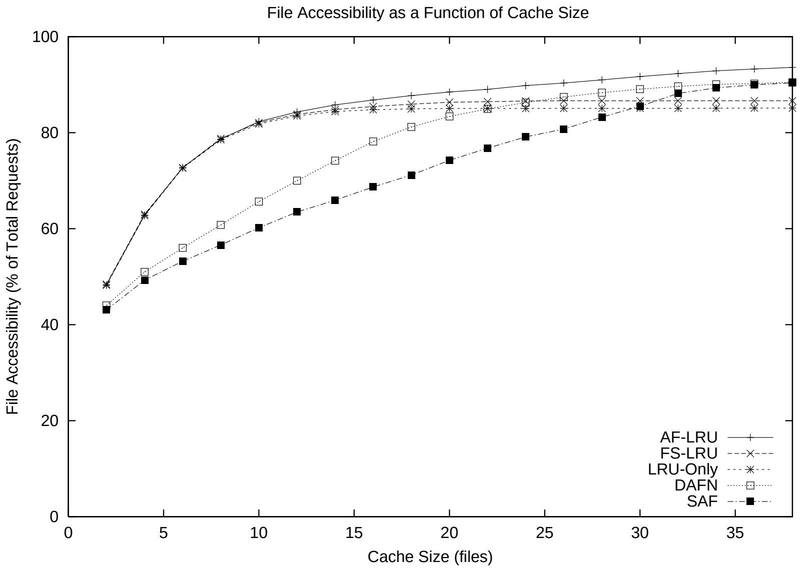

Figure 5.1: File accessibility as a function of cache size

frequencies to files, they will have similar replica sets in their caches, resulting in many duplicated replicas in the system. However, a mobile host can access data items or replicas held by other connected mobile hosts, and it may be more effective to share many kinds of replicas among them. In general, some files may be more popular than others, causing hosts to have similar access frequencies to files. This results in low file accessibility for SAF.

DAFN tries to remove duplicated file replicas between two neighbours from the two caches. It deals with two hosts at a time, and so cannot guarantee to remove all the duplicated replicas between two neighbouring hosts once it finishes replica allocation. As a result, the caches in the system are not effectively utilized either. Nevertheless, DAFN produces better file accessibility than SAF.

The three on-demand policies, AF-LRU, FS-LRU, and LRU-Only, are better than both SAF and DAFN in terms of file accessibility when the cache size is small, since they can respond to the temporal locality in the workload. In particular, AF-LRU, FS-LRU, and LRU-Only have the same performance when the cache size is smaller than 8 files because the cache can be easily filled up, and thus, their performance relies only on the LRU replacement policy to manage the file replicas. When the cache size increases, empty spaces appear in the caches because of the network partitions and the invalidation broadcasts for maintaining replica consistency. Thus the extra effort used in AF-LRU and FS-LRU helps to fill up the empty space, resulting in increasing file accessibility. The file accessibility of LRU-Only stops increasing when the cache size is larger than 16 files because the cache may not be full of replicas.

FS-LRU outperforms LRU-Only by quickly filling up the cache space. Because a file is replicated along the route, in FS-LRU, many duplicate replicas are created between the neighbours. Therefore, cache space may not be effectively utilized in FS-LRU.

AF-LRU is the best policy among the five policies at all cache sizes. When the cache size is small, it responds to the temporal locality by using LRU replacement; when the cache size is large, the empty space in the cache can be utilized by periodically replicating file according to the file access frequencies. Therefore, AF-LRU outperforms the other policies.

It can also be seen in Figure 5.1 that when the cache size is large enough, SAF and DAFN are better than FS-LRU and LRU-Only. This is because SAF and DAFN can easily fill up the cache space. Although FS-LRU can fill up the cache space as well, it creates too many duplicate file replicas between neighbours to improve the file accessibility at large cache size.

Figure 5.2 shows the impact of file updates on the performance of the five replica management policies in terms of file accessibility. As the percentage of write operations increases, file accessibility decreases for the five replica management policies. There are two groupings shown in Figure 5.2, on-demand policies and periodic policies, because write operations affect the on-demand policies, AF-LRU, FS-LRU and LRU-Only, more severely than the periodic policies, SAF and DAFN.

A write operation to a file or its replica will cause an invalidation broadcast. The hosts which receive the invalidation message remove the stale file replica (if it exists) from their caches, causing empty space in their caches. Thus, the caches may not be fully utilized. Since LRU-Only causes file replicas to be created only when

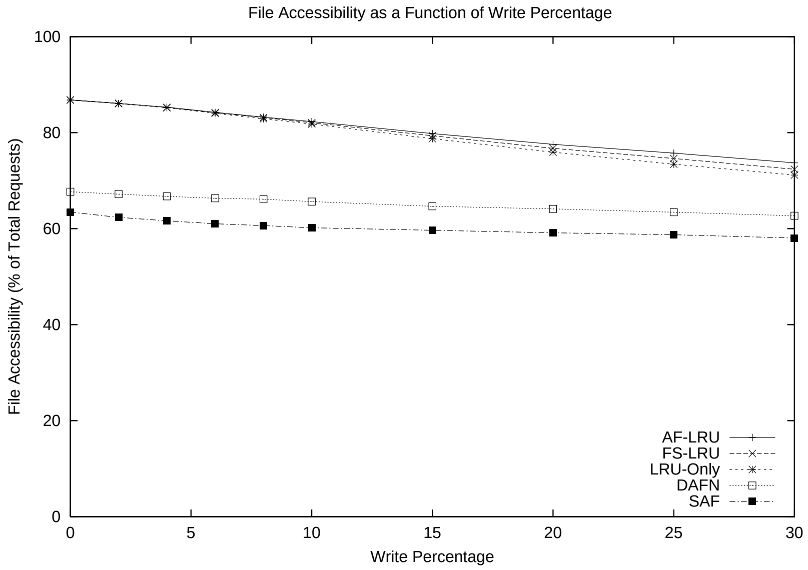

Figure 5.2: File accessibility as a function of write percentage

a host successfully accesses a file, it suffers most from file updates among the five policies. Although there are other ways to create file replicas in AF-LRU and FS-LRU, they are used only when there is enough empty space in the cache. Therefore, file updates affect FS-LRU and AF-LRU more severely than they affect SAF and DAFN, in which file replicas are allocated periodically to fill up the empty cache space.

The effect of the radio range of the mobile hosts on the file accessibility of the five policies is shown in Figure 5.3. For all five policies, file accessibility quickly increases as the radio range increases. When the radio range is very small, every policy gives similar very low file accessibility because the number of connected mobile hosts is very small. Thus mobile hosts have little chance to help each other. When the radio range is very large, file accessibility reaches 100% for all five policies because the network never gets partitioned. A mobile host can always access the files it needs. The file accessibility of the on-demand policies, FS-LRU, AF-LRU and LRU-Only, increases more quickly than that of SAF and DAFN as the radio range increases, causing two groupings to appear in Figure 5.3, because FS-LRU, AF-LRU and LRU-Only can take advantage of the characteristics of concentration and temporal locality

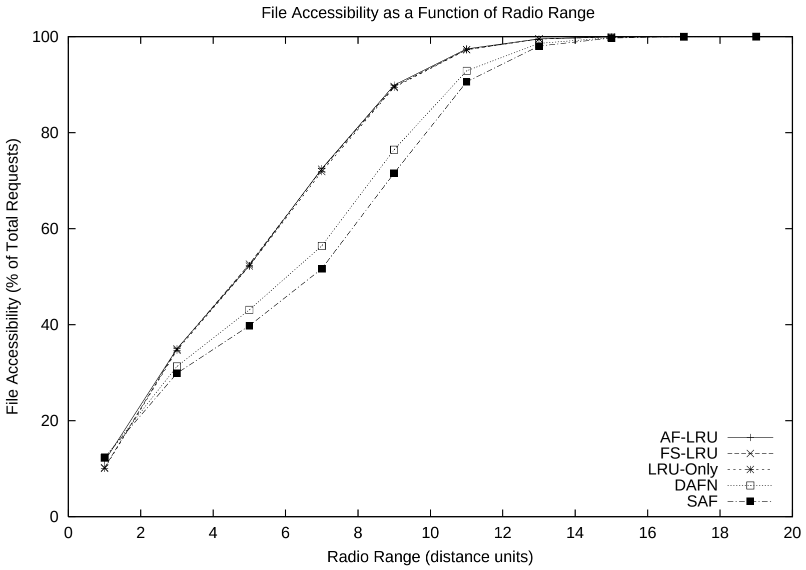

Figure 5.3: File accessibility as a function of the radio range of mobile hosts

in the workload.

Figure 5.4 shows the effect of the allocation period on the file accessibility of SAF, DAFN, and AF-LRU policies. Since FS-LRU and LRU-Only do not cause hosts to periodically allocate file replicas, they are not shown in Figure 5.4.

The allocation period has little effect on the three policies examined. AF-LRU does not cause file replicas to be allocated if there is not enough empty space in the cache at allocation time. Thus the allocation period has almost no effect on the file accessibility of AF-LRU. The file accessibility of SAF decreases slightly as the allocation period increases because invalidating stale file replicas caused by file updates leaves empty space in the caches. Contrary to the intuitive expectation, increasing the allocation period up to a certain point (smaller than 1280 time units in this experiment) actually slightly increases the file accessibility of DAFN. The same result was reported in [29]. When the allocation period is large enough, invalidating stale replicas cause empty space to exist in the cache, which means a small number of file replicas in the network. Therefore, file accessibility decreases.

Since file accessibility is not sensitive to the allocation period, the allocation period should be set to a relatively large value to reduce the network load.

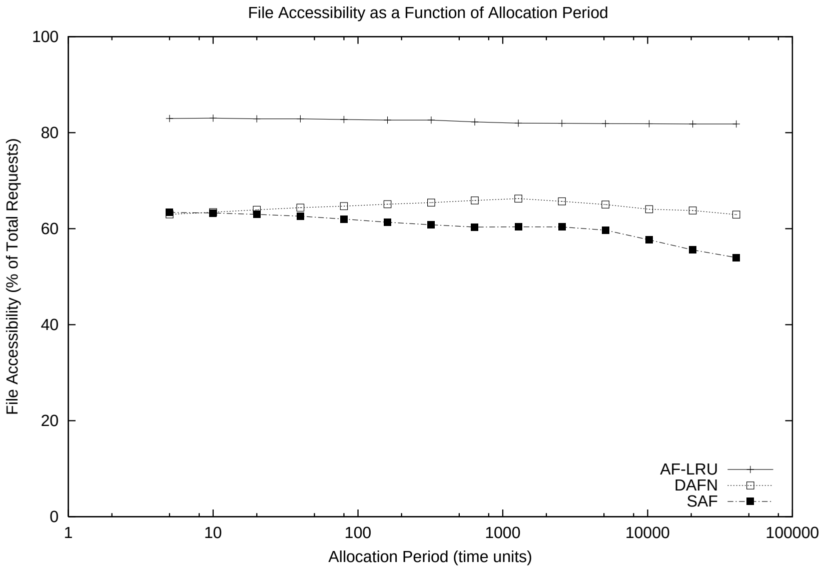

Figure 5.4: File accessibility as a function of allocation period

The results show that AF-LRU outperforms the other policies in terms of file accessibility under different cache sizes, radio ranges, write percentages in the workload, and allocation periods.

5.2.2 Fresh Access

File updates result in stale file replicas in the network. Although the replica management policy is not responsible for the consistency of file replicas, the distribution of file replicas in the network as a result of file replica management affects the effectiveness of the replica consistency management policy. Thus different file replica management policies will result in different degrees of file replica consistency in terms of fresh access under the same replica consistency management policy.

Fresh access is defined as the percentage of successful accesses to the most up-to-date file replicas in the whole workload. Figure 5.5 shows the effect of cache size on the fresh access of the five policies. The two groupings in Figure 5.5 represent the on-demand policies, AF-LRU, FS-LRU, and LRU-Only, and the periodic policies, SAF and DAFN. The on-demand policies outperform the periodic policies.

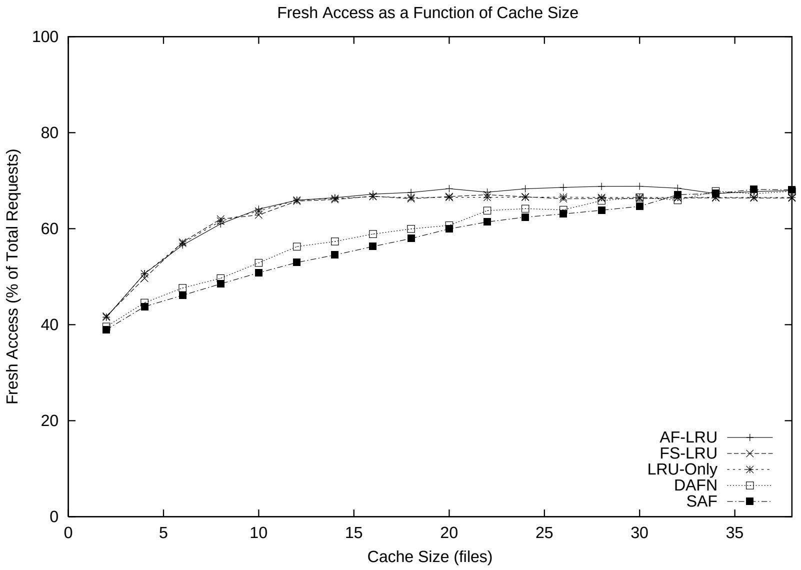

Figure 5.5: Fresh access as a function of cache size

Fresh access increases at first and then stops increasing at certain points for the five policies as the cache size increases. The reason is that increasing the cache size increases the file accessibility, and the fresh access increases as well. The rate of increase for all five policies is lower than the rate of increase of file accessibility shown in Figure 5.1. This is because increasing the cache size results in more file replicas in the network, which in turn makes it more difficult to maintain replica consistency. The difference between the five policies is not large. This means that the replica management policy has little effect on maintaining file replica consistency.

Figure 5.6 shows the effect of the write percentage on the fresh access of the five replica management policies. Again, there are two groupings in Figure 5.6 reflecting the difference between the on-demand policies and the periodic policies. Not surprisingly, increasing the write percentage in the workload results in decreasing fresh access for all five policies considered.

To improve file accessibility, the replica management policy tries to spread file replicas into the network evenly to alleviate the problem of network partitions. This has a negative effect on file replica consistency. Therefore, the fresh access of FS-LRU, AF-LRU, and LRU-Only drops more quickly than that of SAF and DAFN as

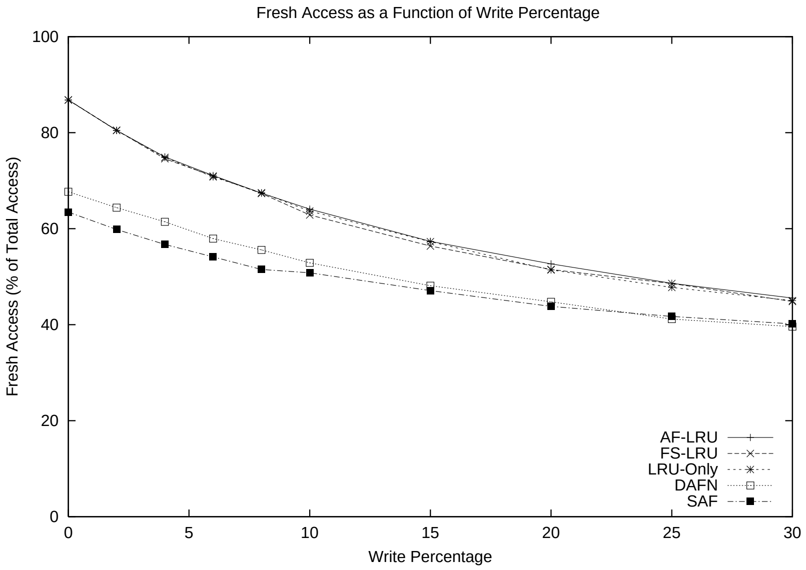

Figure 5.6: Fresh access as a function of the write percentage in the workload

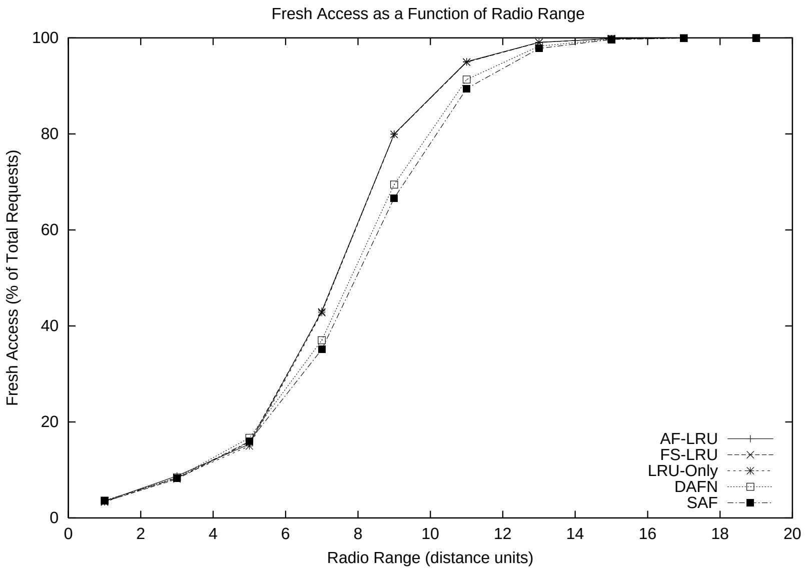

Figure 5.7: Fresh access as a function of the radio range of mobile hosts

the write percentage in the workload increases.

The radio range of the mobile hosts affects fresh access significantly. Large radio range means the network rarely gets partitioned and the stale file replicas can be easily invalidated by the invalidation broadcasts. Therefore, fresh access reaches

100% for all replica management policies when the radio range is huge. Figure 5.7 shows fresh access as a function of the radio range of the mobile hosts. The radio range of the mobile hosts affects fresh access in the same way for all five replica management policies. The fresh access for on-demand policies increase more quickly than that for periodic policies as the radio range increases, causing two groupings to appear in Figure 5.7.

Figure 5.8 shows the impact of the allocation period on fresh access for AF-LRU, SAF, and DAFN. Again, fresh access for the three policies is only very slightly affected by the allocation period. The difference in fresh access among the policies is caused by the difference in file accessibility among the policies since higher file accessibility will likely result in higher fresh access.

In general, a file replica management policy has little effect on the consistency of file replicas. High file accessibility likely results in high fresh access. Therefore, AF-LRU provides better fresh access than the other policies.

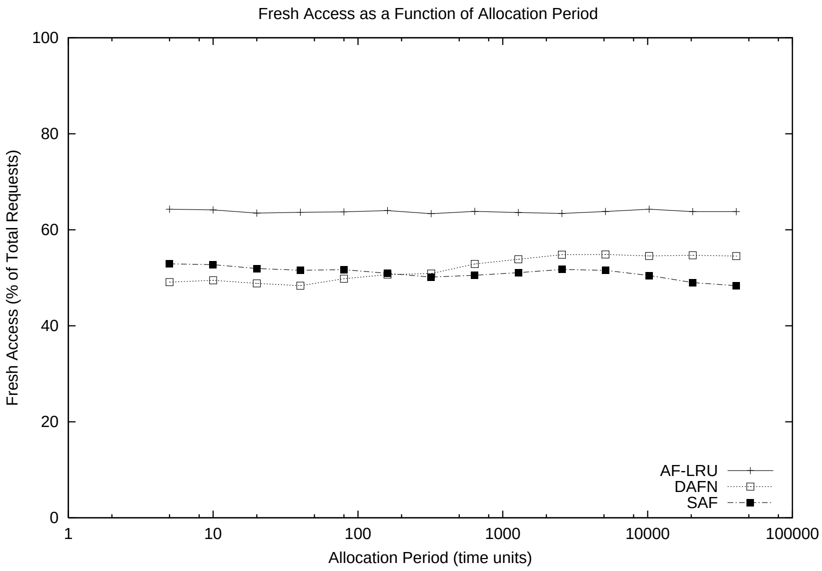

Figure 5.8: Fresh access as a function of allocation period

5.2.3 Average Hops per Access

One of the goals of file replication is to reduce the response time of file access by allocating file replicas to the place where they are likely to be used. In a mobile ad hoc network, multi-hop communication is the common communication pattern. Decreasing hop count on a communication route will decrease the response time. This is achieved by allocating file replicas close to the hosts who need to access them.

The average hops per access (described in Section 3.4.1) is used to reflect the response time for different file replica management policies. A small average hops per access is desirable for a replica management policy since it means a host can quickly get the file replica it needs and little network load is imposed on the network.

Figure 5.9 shows the effect of cache size on the average hops per access for the five replica management policies. Increasing the cache size decreases the average hops per access for all policies because a large cache size allows a host to allocate more file replicas into its cache. Thus, a host has more chances to get its requested files from its own cache or from a nearby host's cache. When the cache size is small,

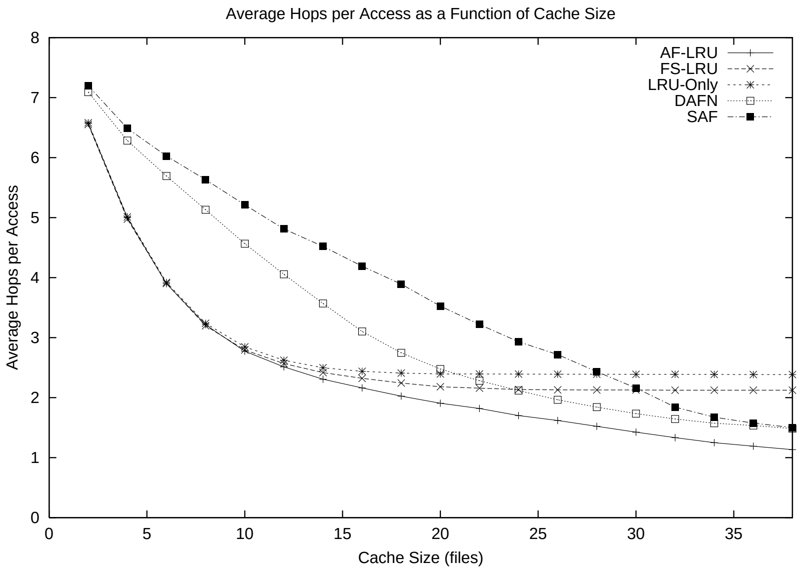

Figure 5.9: Average hops per access as a function of cache size

a host has to get most of the requested files from the other hosts which may be far away, resulting in a high average hops per access.

The average hops per access has a relationship with file accessibility. Higher file accessibility implies that there are more file replicas in the network, and hence a host has more chances to get a file replica from a nearby host resulting in a lower average hops per access. Therefore, Figure 5.9 is similar to the "inverse" of Figure 5.1. For the same reason, such a relationship can also be seen in other pairs of figures: Figure 5.2 and Figure 5.10, Figure 5.3 and Figure 5.11, and Figure 5.4 and Figure 5.12.

Figure 5.10 shows the effect of the write percentage on the average hops per access for the five policies. Increasing the write percentage increases the average hops per access because file updates cause some hosts to invalidate stale file replicas. Thus the number of file replicas in the network decreases, and the chance for a host to get a file replica from a nearby host becomes small. Therefore, the average hops per access for all policies increases as the write percentage increases in the workload.

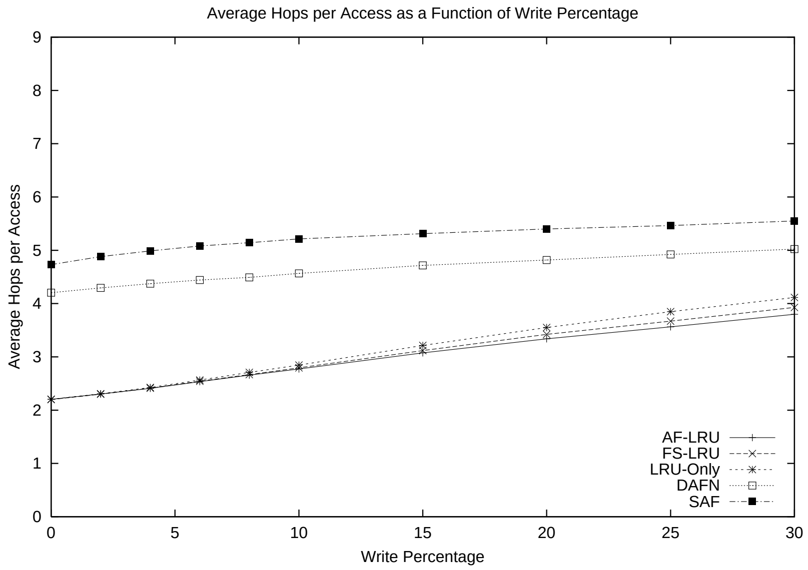

Figure 5.10: Average hops per access as a function of write percentage

The write percentage affects the on-demand policies, FA-LRU, FS-LRU, and LRU-Only, more severely than the periodic policies, SAF and DAFN. For the two periodic policies, the increase rates are the same. For the three on-demand policies, the increase rate of AF-LRU is slightly lower than that of FS-LRU and LRU-Only because the periodic allocation method used in AF-LRU reduces the effect of stale replica invalidation.

Figure 5.11 shows the effect of the radio range of the mobile hosts on the average hops per access. For all five policies, increasing the radio range decreases the average hops per access dramatically. This is because a large radio range means the network rarely gets partitioned and the number of multi-hop communication paths becomes small. The average hops per access for the on-demand policies drops more quickly than that for the periodic policies as the radio range increases, resulting in two groupings in Figure 5.11.

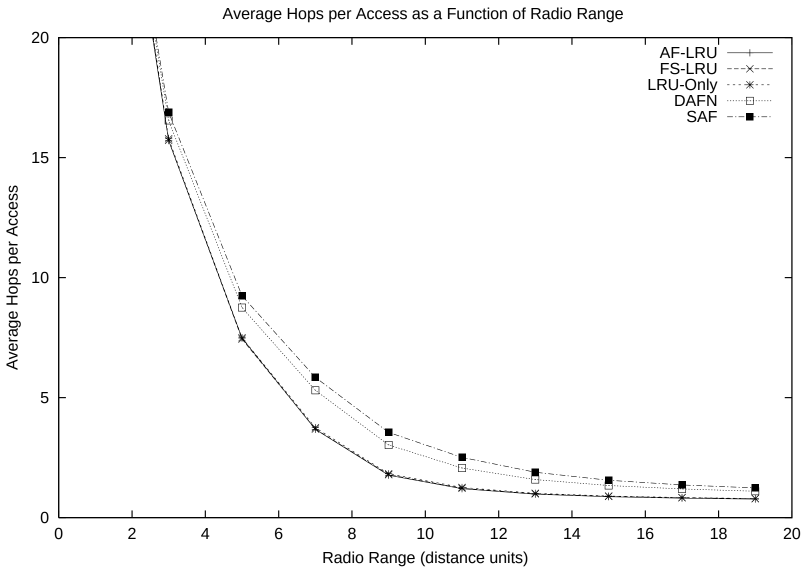

Figure 5.11: Average hops per access as a function of radio range

Figure 5.12 shows the impact of the allocation period on the average hops per access for the three policies, AF-LRU, SAF, and DAFN. The allocation period has little effect on AF-LRU because most of file replicas are allocated when mobile hosts successfully access them; periodically allocating file replicas is used to utilize the cache space only when the cache is not full. In DAFN and SAF, the periodic allocation is used to fill up the empty space caused by invalidating stale file replicas, and a large allocation period makes the contents of hosts' caches remain unchanged for a long time. Therefore, the hop count for a host to get a file increases. Since some duplicate file replicas between neighbours are removed in DAFN, the number of different kinds of file replicas increases. Thus DAFN provides better response time (average hops per access) than SAF does.

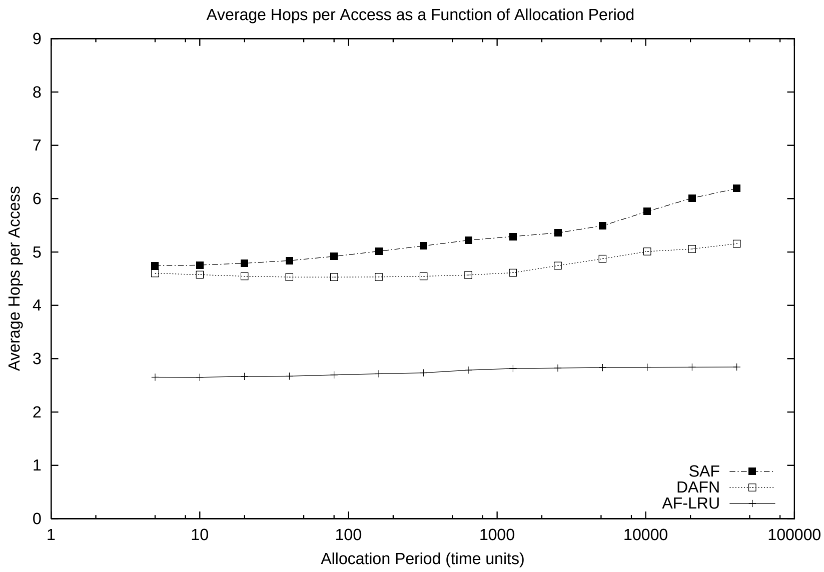

Figure 5.12: Average hops per access as a function of allocation period

From the viewpoint of the average hops per access, AF-LRU clearly outperforms the other policies under all situations discussed. This is because the LRU replacement policy can capitalize on the temporal locality in the workload to provide high file accessibility, and the cache space can be further utilized by the periodic allocation method when there are empty spaces in the cache caused by invalidating stale replicas. Although AF-LRU, FS-LRU, and LRU-Only significantly outperform DAFN and SAF, the performance differences among AF-LRU, FS-LRU, and LRU-Only are very small in terms of average hops per access.

5.2.4 Network Load

When bandwidth is limited, it is important that it be used wisely. To reflect the network load imposed by a file replica management policy, a count of total hops is used. This includes the hop count for mobile hosts accessing requested files and the hop count for transferring files for allocating file replicas. For example, at an allocation time under the policy SAF, DAFN, or AF-LRU, a host may have to get a file from a far away host to replicate it into its cache. Since there is no periodic allocation used in FS-LRU and LRU-Only to allocate file replicas, FS-LRU and LRU-Only should impose a small amount of network load only for accessing files. Because DAFN periodically allocates file replicas according to access frequencies first, then removes duplicate replicas between neighbours, it should impose a large amount of network load.

Figure 5.13 shows the total hops as a function of cache size for the five policies. As expected, DAFN gives the highest total hops of the five policies. As the cache size increases, the total hops of DAFN increases at first, then decreases once the cache size exceeds a certain value. This is because the number of hops used to remove the duplicate replicas between neighbours increases as the cache size increases. As the cache size increases further, the caches can hold most of replicas of files in the network, which means it is impossible to remove all the duplicate replicas between neighbours while keeping the caches full of replicas. When this happens, DAFN leaves most duplicate replicas unchanged, and performs a small number of operations for removing replica duplicates. Thus, the number of hops used to remove duplicate replicas decreases. In SAF, a large cache size means a large number of replicas are allocated in the cache. Therefore, the number of total hops increases as the cache size increases.

AF-LRU, FS-LRU, and LRU-Only have similar curves when the cache size is small (less than 12 files). As the cache size increases, the number of total hops for the three policies increases at first, then decreases. Because a small cache size causes a small number of replicas in the network, which in turn, causes low file accessibility, this results in a small number of total hops. As the cache size increases, file accessibility increases, and the number of total hops for the three policies increases as well. As the cache size gets larger, hosts can hold more file replicas in their caches, and so there are more chances for a host to get a file from a nearby host. Therefore, the number of total hops starts decreasing.

When the cache size exceeds 12 files, the differences among the three policies becomes large. The differences are caused by the different ways the policies deal with the empty cache space caused by invalidating stale replicas. In LRU-Only, no special way is used to fill up the empty cache space. Thus, the cache space is not utilized effectively. In FS-LRU, hosts are allowed to replicate files as they deliver the files as routers if there is enough empty space in their caches. In this way, many duplicate replicas may be created, and so the cache space is not utilized effectively either. Therefore, the number of total hops for LRU-Only and FS-LRU decreases slowly and gradually stops decreasing as the cache size increases. Nevertheless, FS-LRU provides better utilization of cache space than LRU-Only, and uses a smaller number of total hops than LRU-Only does. In AF-LRU, a periodic allocation method is used to fill up the empty cache space. Thus the cache space is utilized effectively, which

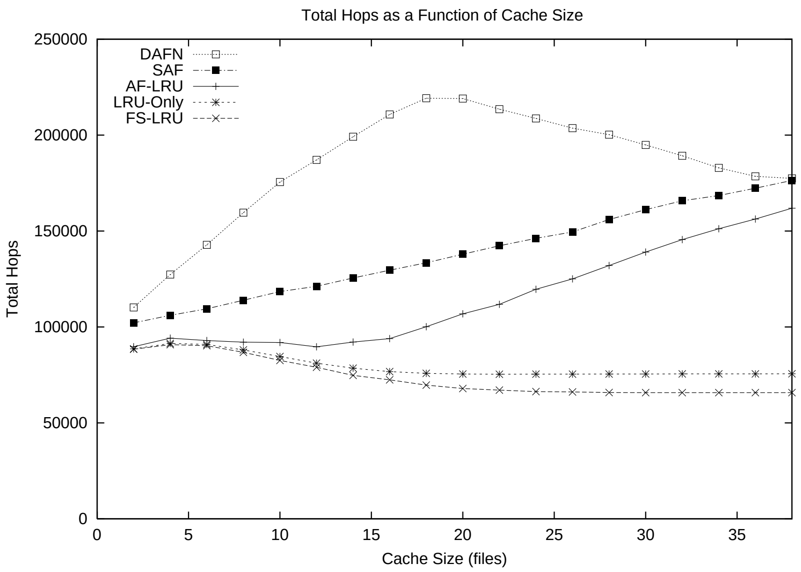

Figure 5.13: Total hops as a function of cache size

is reflected in high file accessibility for AF-LRU (as shown in Figure 5.1). But such a utilization comes with a cost of a number of extra hops for periodic allocation. A large cache size means a large amount of empty cache space and a large number of extra hops for periodically allocating file replicas. Therefore, the number of total hops for AF-LRU starts increasing again as the cache size increases further.

The radio range of the mobile hosts has a similar effect as cache size on the total hops for the five policies. Figure 5.14 shows the effect of the radio range on the total hops for the five policies. As the radio range increases, the total hops for all policies increases at first, then drops for all policies but DAFN, when the radio range exceeds a certain value. When the radio range is small, hosts may not get requested files because of network partitions. Thus the number of total hops for accessing and allocating file replicas is small. Increasing the radio range reduces the frequency of network partitioning. Thus hosts can successfully access most requested files causing a large number of total hops for accessing and allocating file replicas. Further increasing the radio range makes a file transfer require fewer hops. Thus the total hops decreases. For DAFN, a large radio range makes it easy to get the required file replicas to remove duplicate file replicas between neighbours. Thus, there are more file replica allocating activities involved. Therefore, the total hops keeps increasing as the radio range increases further.

The write percentage in the workload affects the total hops for all five policies as well. Figure 5.15 shows the effect of write percentage in the workload on the total hops for the five policies. Increasing the write percentage in the workload causes the number of total hops to increase for all policies. File updates cause mobile hosts to invalidate stale replicas in their caches, which in turn, causes empty cache space to appear. Thus extra hops may be needed for hosts to access requested file replicas. Therefore, the number of total hops increases as the write percentage increases for all policies. In DAFN, SAF and AF-LRU, periodic allocation methods are used. A larger amount of empty cache space causes more replicas to be allocated, and more hops are need for transferring the replicas. Therefore, the write percentage affects DAFN, SAF and AF-LRU more severely than LRU-Only and FS-LRU, in which

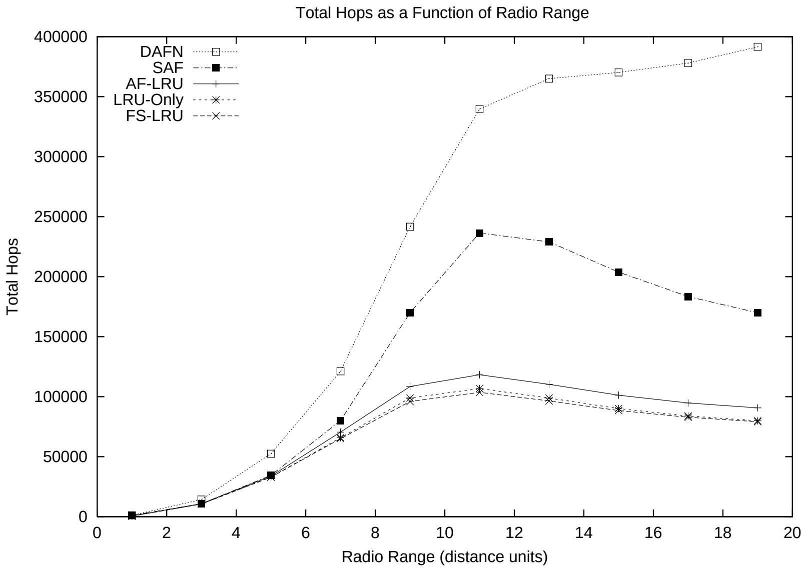

Figure 5.14: Total hops as a function of radio range

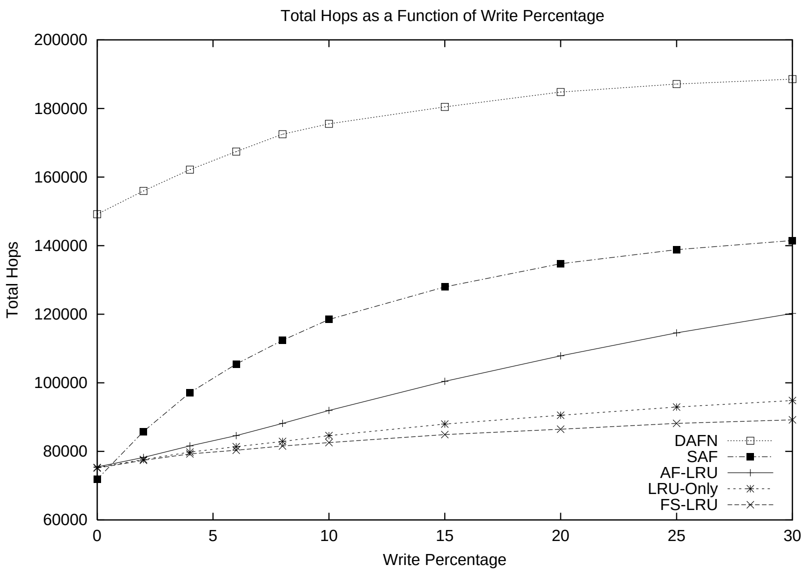

Figure 5.15: Total hops as a function of the write percentage in the workload

hosts do not allocate replicas periodically. In FS-LRU, a host is allowed to replicate the files when the host delivers them as a router. Thus the empty cache spaces can be filled up with new replicas without extra hops. Therefore, the write percentage has the least effect on FS-LRU among the five policies. In LRU-Only, no extra effort is used to fill the empty cache space. Thus the number of replicas in the network decreases as the write percentage increases. Hosts may need more hops to access requested files causing the number of total hops to increase.

The allocation period dramatically affects the total hops for DAFN because of the effort expended by DAFN in removing replica duplication between neighbours. Figure 5.16 shows the total hops for DAFN, SAF, and AF-LRU. For DAFN, a small allocation period cause a huge number of total hops, and slightly increasing the allocation period decrease the total hops significantly. For AF-LRU and SAF, the total hops decrease very slightly as the allocation period increases. Since the allocation period has little effect on file accessibility for the three policies (as shown in Figure 5.4), a relatively long value of allocation period is preferred. For DAFN, the allocation period should be long enough to reduce the network load.

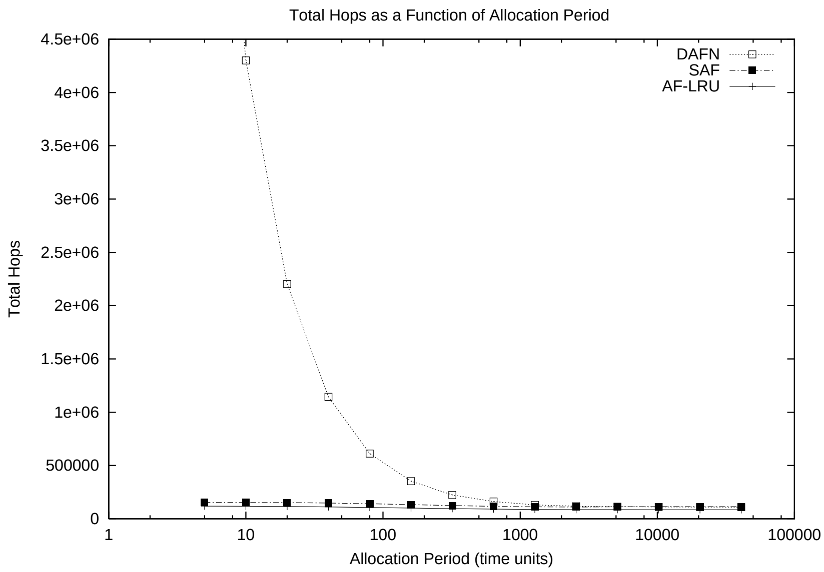

Figure 5.16: Total hops as a function of allocation period

5.3 Summary

Five file replica management policies were examined: DAFN, SAF, AF-LRU, FS-LRU, and LRU-Only. The effectiveness of the five policies, in terms of file accessibility, fresh access, average hops per access and total hops, was investigated through simulation experiments. Four factors, cache size, write percentage in the workload, radio range and allocation period, were varied for the experiments.

The results show that the three on-demand allocation policies, AF-LRU, FS-LRU, and LRU-Only, outperformed the two periodic allocation policies, DAFN and SAF, in most situations considered. DAFN and SAF can provide higher file accessibility than FS-LRU and LRU-Only only when the cache size is high enough, but DAFN and SAF impose much more network load on the network than FS-LRU and LRU-Only in most situations. The fresh accesses of DAFN and SAF are lower than those of AF-LRU, FS-LRU, and LRU-Only. For DAFN and SAF, the response time measured as the average hops per access is not good either. DAFN and SAF also require mobile hosts to allocate file replicas at the same time. This synchronized allocating could possibly tie up the wireless network for a considerable time. Thus DAFN and SAF may not be good choices for mobile ad hoc networks.

The results also show that there is a "best" cache size for FS-LRU and LRU-Only, beyond which further increasing the cache size will not improve performance. File updates cause some empty spaces in the caches because of invalidating stale file replicas. Lacking other methods to allocate file replicas, LRU-Only cannot utilize the whole cache space. Thus increasing cache size beyond a certain point will not improve the performance of LRU-Only. FS-LRU allows mobile hosts to create file replicas, if necessary, when they relay files on the multi-hop communication paths between the source and destination hosts. Thus, in FS-LRU, the empty spaces in the cache caused by file updates can be filled up. But many duplicate file replicas are created between neighbours. In ad hoc networks, sharing many kinds of replicas between neighbours is very important. Therefore, FS-LRU cannot fully utilize the cache space either.

From the results obtained, AF-LRU appears to be the best replica management policy among the five policies examined. When the cache size is small, on-demand allocation combined with LRU replacement capitalizes on the concentration and temporal locality characteristics in the workload. Thus AF-LRU gives the same high file accessibility as FS-LRU and LRU-Only. When the cache size is large, asynchronous periodic allocation (i.e., every host periodically allocates file replicas according to its own schedule, as opposed to SAF and DAFN, in which every host allocates file replicas at the same time) will fill up the empty cache space caused by stale replica invalidation. Thus AF-LRU provides even better performance than DAFN and SAF. AF-LRU imposes moderate network load, less than DAFN and SAF but higher than FS-LRU and LRU-Only. In general, the combination of on-demand and periodic allocation is a good choice for mobile ad hoc networks.

Chapter 6

Experiments with Replacement Policies

As a part of the replica management policy, a file replica replacement policy could have a definite impact on the performance of a replica management policy. Replica replacement occurs when the cache is full and a new file replica must be brought into the cache. Which replica is chosen to be removed from the cache directly affects the distribution of the file replicas in the network, and therefore affects the system performance.

In this chapter, the performance of different replica replacement policies are examined. The experimental design, including the replica replacement policies examined, the performance metrics used, and the parameter settings for the simulation experiments, is described in Section 6.1. The experimental results are presented and discussed in Section 6.2. A brief summary of the chapter is given in Section 6.3

6.1 Experimental Design

When the cache size and the write percentage in the workload are large, the cache may not be full all the time. To alleviate this problem, the periodic access frequency-based allocation method (used in AF-LRU) and the fast-spread allocation method (used in FS-LRU) are used with on-demand allocation to effectively utilize the cache space. In Chapter 5, the results showed that AF-LRU is the best policy among the five replica management policies considered. Therefore, to evaluate the performance impact of different replica replacement policies, four replica management policies were used in the simulation experiments: AF-LRU, AF-LFU, AF-LAF, and AF-DWF. The operations of AF-LFU, AF-LAF, and AF-DWF are essentially the same as that of AF-LRU (described in Section 3.2) except for the way they deal with replica replacement (described in Section 3.2.1). Since they use the same periodic allocation method, the performance differences among AF-LRU, AF-LFU, AF-LAF, and AF-DWF are entirely due to the different replacement policies.

Another way to evaluate the performance of the different replica replacement policies is to use a combination of the fast-spread allocation method and the replica replacement policy. In this way, four replica management policies could be formed: FS-LRU, FS-LFU, FS-LAF, and FS-DWF. Their operations are the same as FS-LRU (described in Section 3.2) except for the way they deal with replica replacement.

With respect to the performance differences among the different replica replacement policies, the results from the experiments with AF-LRU, AF-LFU, AF-LAF, and AF-DWF should be the same as those from the experiments with FS-LRU, FS-LFU, FS-LAF, and FS-DWF because periodic access frequency-based allocation or fast-spread allocation interact with replica replacement rarely. For example, replica replacement occurring frequently means the cache is full, and so there is no need to replicate files using either the periodic frequency-based allocation or fast-spread allocation. On the other hand, if it is necessary to replicate files using periodic frequency-based allocation or fast-spread allocation because the cache is not full, replica replacement occurs rarely.

6.1.1 Policies Investigated

With periodic access frequency-based allocation, four replica replacement policies, LRU, LFU, LAF, and DWF (described in Section 3.2.1), were examined:

- LRU (Least Recently Used) – LRU is the most commonly used cache replacement policy. It can capitalize on the temporal locality in the workload, and normally provides good performance.

- LFU (Least Frequency Used) – In LFU, a reference count is maintained for each replica. The count is incremented each time the replica is accessed. The replica with the lowest reference count is the least frequently used and is the one chosen for replacement. In traditional file systems, LFU may not perform well because some replicas will build up large counts in a short period of time, and then rarely be replaced, even if they will not be accessed for a long period of time. However, those replicas may be used by the host's neighbours in mobile ad hoc network environments.

- LAF (Lowest Access Frequency) – In LAF, the replica with the lowest access frequency (which is assumed to be known in advance) is replaced. LAF looks at the long term effect since the replica with the lowest access frequency will likely not be accessed in a short time.

- DWF (Dynamic Write Frequency) – In DWF, the replica with the highest write access frequency is replaced. Frequently updated replicas are not good candidates for caching since they are more likely to be stale than those that are updated infrequently.

6.1.2 Performance Metrics and Factors

The following performance metrics are used for evaluating different replacement policies: file accessibility, fresh access, and average hops per access (described in Section 3.4.1). Three factors are used to show their impact on the performance of a replacement policy: cache size, write percentage in the workload, and radio range (described in Section 3.4.2).

6.1.3 Parameter Settings

The parameters used in these experiments are shown in Table 6.1. The parameter settings and the reasons for the setting are essentially the same as in the evaluation of replica management policies in Section 5.1.3.

The reconciliation policy was set to be CH (Closest Host) (see Section 3.3) since

Table 6.1: Factors and levels for the simulation experiments

Factors	Levels
Replacement policies	LRU, LFU, LAF, DWF
Reconciliation policy	CH
Reconciliation period (time unit)	20
Cache size	2 to 38 files, in steps of 2 files
Write percent in workload	2%, 4%, 6%, 8%, 10%, 15%, 20%, 25%, 30%
Radio range (distance unit)	1, 3, 5, 7, 9, 11, 13, 15, 17, 19
Allocation period (time unit)	512

it provides better performance than FO (File Owner) does in most situations considered in this study (as shown in Section 7.2). The preliminary experiments showed that the reconciliation period (see Section 3.4.2) should be set to be a relatively short value to improve replica consistency. The value was set to be 20 time units, which means that every 20 time units, a host will initiate reconciliation processes to reconcile with the partners chosen according to the CH policy.

A large range of cache sizes was used to examine the impact of cache size on the performance of replacement policies. To investigate the effect of file updates on the performance of replica replacement, the percentage of write operations in the workload was varied from 0% to 30%. Radio range is an important parameter in a mobile ad hoc network, since it affects the frequency of network partitioning which is the main cause of low file accessibility. The preliminary experiments showed that setting the radio range to be 19 distance units in a flat 50x50 network area (see Section 4.3.1) causes few network partitions throughout the simulation time. Thus the biggest value of radio range examined was 19 distance units.

Experiments were run by varying the value of one parameter at a time while keeping the others fixed at specific values. Table 6.2 shows the values of parameters when they were not changed.

Table 6.2: Fixed values for parameters when they were not changed

Parameter	Value
Cache size	10 files
Radio range	8 distance units
Write percent in workload	10%

6.2 Experimental Results

Three sets of experiments were conducted by varying different parameters to show how these factors affect the performance of replica replacement policies by using the AF-LRU, AF-LFU, AF-LAF and AF-DWF replica management policies. The results are discussed in the following three sections.

6.2.1 File Accessibility

Figure 6.1 shows the performance of the four replacement policies in terms of file accessibility under different cache sizes. When the cache size is very small, the choice of a replica to be replaced is limited, regardless of which file replica replacement policy is used. Thus the four policies provide very similar file accessibility. When the cache size is very large, replica replacement rarely occurs because the cache can hold almost all replicas of all files. Therefore, the four policies give the same performance.

When the cache is of moderate size, LRU gives the highest file accessibility because it can capitalize on the temporal locality in the workload. LFU performs well because the temporal locality will cause the most recently used replica to have a higher reference count. Therefore, LFU can also benefit from the temporal locality in the workload. LAF gives the worst performance among the four replacement policies. Because LAF removes the replica with the lowest access frequency, and hosts in the network have similar file access frequencies to some popular files in the network, LAF gradually causes hosts to have similar sets of replicas in their caches. This causes a large number of replicas of popular files (resulting in many

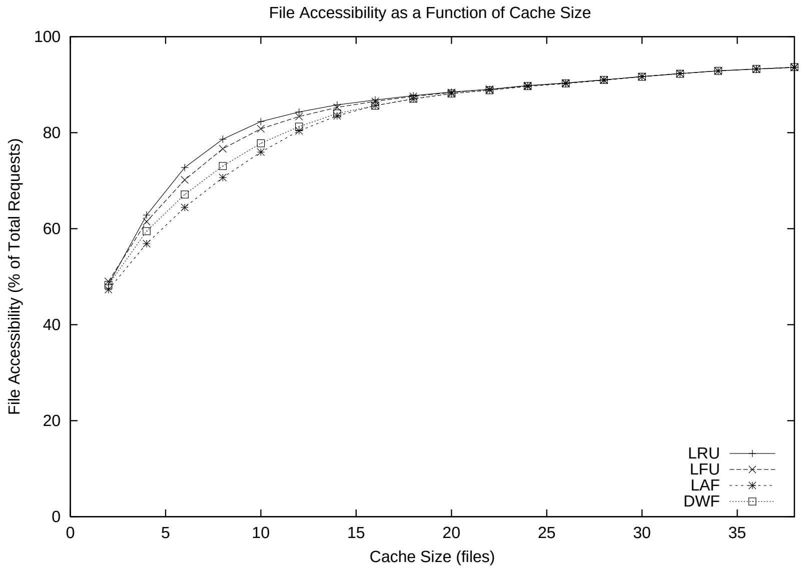

Figure 6.1: File accessibility as a function of cache size

duplicate replicas between neighbours) and a small number of replicas of other files in the network. Access requests to the files with small number of replicas will likely fail, resulting in low file accessibility. Therefore, sharing replicas of different files is very important in a mobile ad hoc network. In DWF, the replica with the highest write access count is replaced because the frequently updated file replicas are more likely to be stale. But frequently updated files are likely to be accessed frequently. Therefore, DWF gives lower file accessibility than LRU and LFU. Nonetheless, file accessibility is similar for the four policies under different cache sizes, the biggest difference is only 10%.

Figure 6.2 shows the effect of the write percentage in the workload on file accessibility for the four replica replacement policies. As the write percentage in the workload increases, file accessibility for LRU, LFU, and DWF decreases slightly because of replica invalidation caused by file updates. When the write percentage is large, frequent replica invalidations cause many empty spaces in the cache, and so replica replacement occurs infrequently. This makes the four replacement policies perform similarly. When the write percentage gets smaller, replica replacement occurs frequently and the differences among the policies become larger.

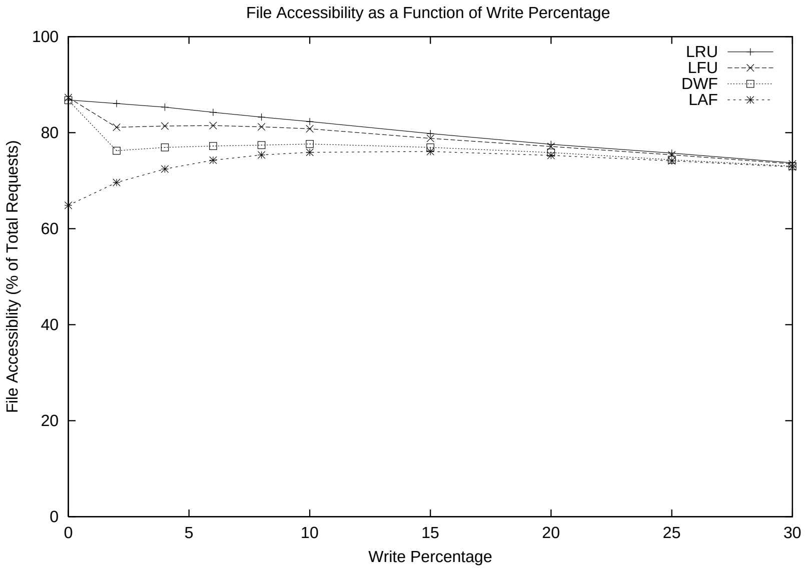

Figure 6.2: File accessibility as a function of write percentage

File accessibility for LAF actually increases as the write percentage in the workload increases. This is because LAF has a tendency to make every host have a similar set of replicas because of the concentration characteristic in the workload, resulting in many replica duplications among the mobile hosts. Replica invalidations caused by write access reduce the number of replica replacements performed, and so LAF's tendency to make similar sets of replicas among the mobile hosts is reduced. As a result, the number of replicas of different files among the mobile hosts increases, resulting in increasing file accessibility. Nevertheless, LAF gives the worst performance in terms of file accessibility because of too many duplicate replicas among mobile hosts.

The effect of the radio range on the performance of the four replacement policies in terms of file accessibility is shown in Figure 6.3. Increasing the radio range increases file accessibility for all replacement policies. When the radio range is small, a host may not be able to access the requested files successfully because of network partitions. Hence, replica replacement occurs rarely. When the radio range is large, which replica should be replaced is not important anymore for file accessibility because a host can get the requested files from the other hosts. Therefore file accessibility for the four replacement policies is the same at small or large radio ranges. The small difference of file accessibility among the replacement policies at moderate radio range is due to the same reasons described in the previous paragraphs.

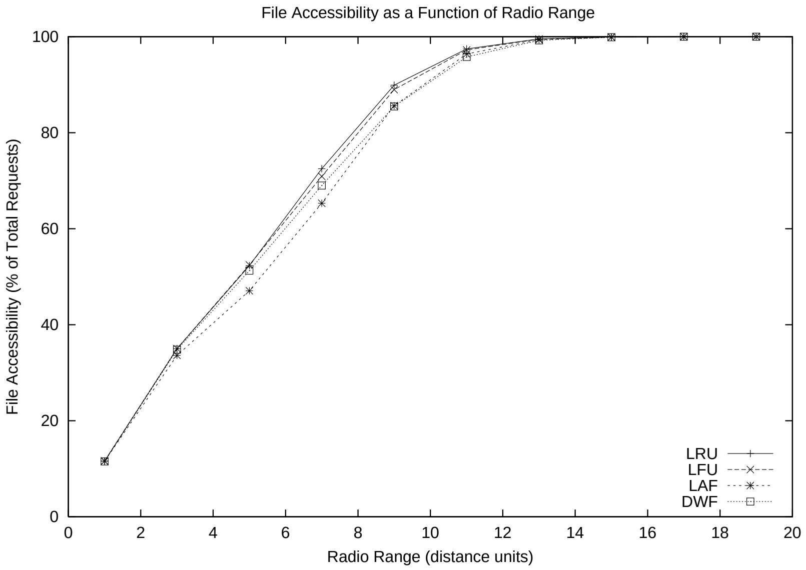

Figure 6.3: File accessibility as a function of radio range

Although different replica replacement policies result in different performance in terms of file accessibility, the performance difference among the four policies is small. By capitalizing on the temporal locality in the workload, LRU slightly outperforms LFU, LAF, and DWF in most situations discussed. Since temporal locality is a common characteristic of most workloads, LRU appears to be a better choice than the other policies considered.

6.2.2 Fresh Access

Although a replacement policy is not responsible for maintaining replica consistency, it affects the degree of consistency among the file replicas. Fresh access can reflect the effect of replica replacement on replica consistency.

Fresh access is defined as the percentage of successful accesses to the most up-to-date file replicas in the whole system. Figure 6.4 shows fresh access as a function of cache size. Increasing cache size increases fresh access for all replacement policies because file accessibility increases. DWF performs worst among the four policies. DWF tries to replace the frequently updated file replicas to improve fresh access. But decreasing the number of frequently updated replicas also decreases the number of most up-to-date versions of those replicas, and so decreases the chances for the hosts to access the most up-to-date replicas. The result is that the fresh access is actually lower than that of the other policies. LRU gives better performance than LFU and LAF, but the difference among them is small.

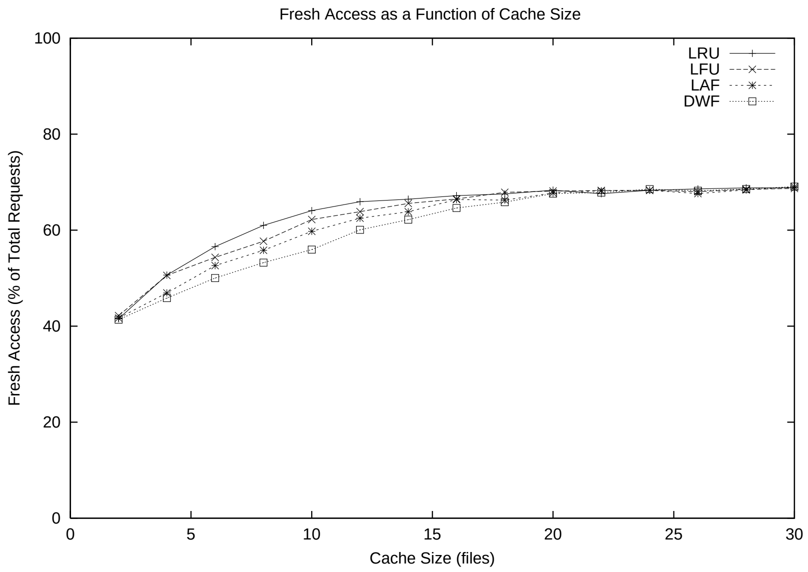

Figure 6.4: Fresh access as a function of cache size

Figure 6.5 shows the effect of the write percentage in the workload on fresh access for the four policies. As the write percentage in the workload increases, the fresh access decreases for all policies. When the write percentage in the workload is small, DWF is affected more severely than the other policies for the same reason described above. When the write percentage is large, the difference among the four policies becomes very small. This implies that replica replacement has little effect on replica consistency.

The effect of the radio range on fresh access is shown in Figure 6.6. Not surprisingly, as the radio range increases, fresh access increases quickly for all policies. A

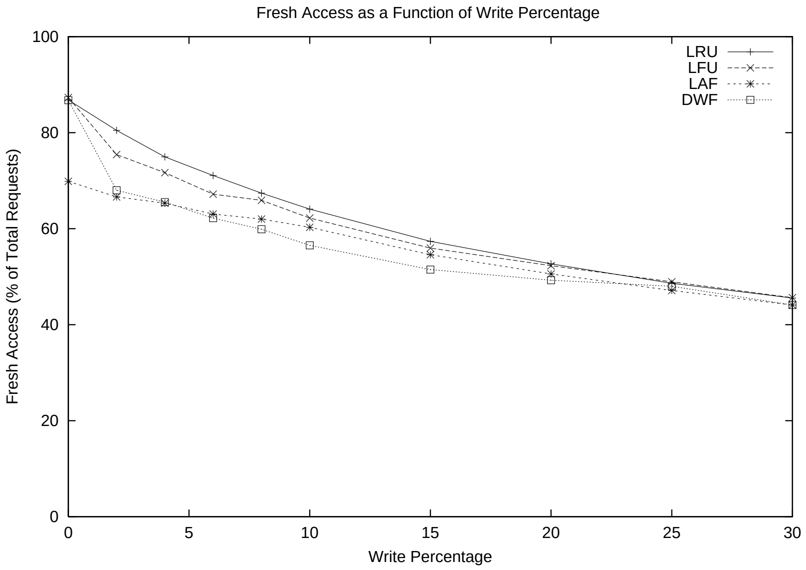

Figure 6.5: Fresh access as a function of write percentage

high radio range makes network partitioning occur infrequently and so stale replicas can be easily invalidated. Therefore, the fresh access increases as the radio range increases. Again, the differences among the four policies are very small, which implies that replica replacement has little effect on replica consistency.

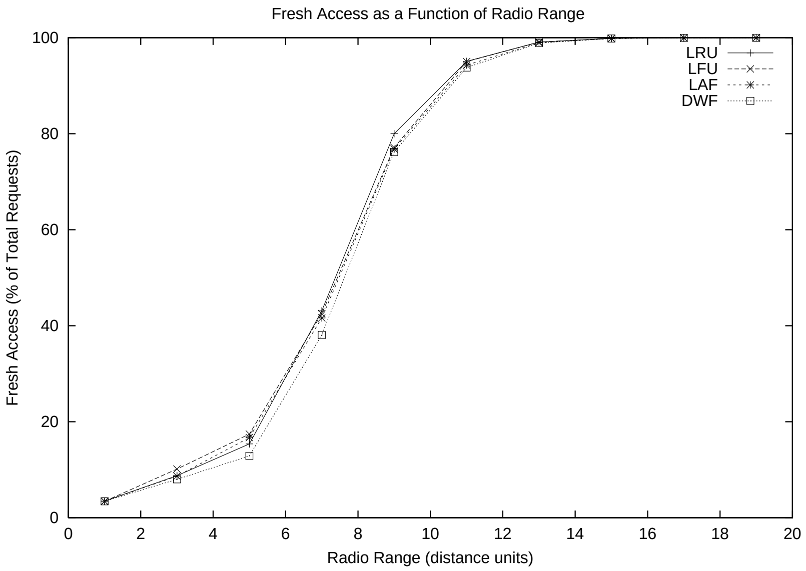

Figure 6.6: Fresh access as a function of radio range

In most situations considered, DWF performs slightly worse than the other policies. The primary reason is that replacing frequently updated files may result in decreasing the number of the most up-to-date replicas of those files in the network, which in turn, degrades the performance of the reconciliation process. The results show that the performance difference in terms of fresh access is very small among LRU, LFU, LAF and DWF.

6.2.3 Average Hops per Access

The average hops per access reflects the response time for file accesses. It is defined as the ratio of the total number of hops for file accesses, which includes both successful accesses and unsuccessful accesses, to the total number of access requests in the workload. The hop count for an unsuccessful access is defined as the network diameter plus 1 (see Section 3.4.1) since a host needs to wait for a certain time to know the access has failed. A small average hop count means a host can get the requested files quickly, and also implies a small number of unsuccessful accesses.

Figure 6.7 shows the effect of cache size on the average hops per access. Increasing the cache size quickly decreases the average hops per access for all policies because a large cache size allows a host to hold more file replicas in its cache. Thus a host has more chances to get the requested file from a nearby host. Therefore, the average hops per access decreases as cache size increases. LRU appears to have the lowest average hops per access among the other policies considered, in general, the difference of the average hops per access among the four polices is small.

Higher file accessibility implies that there are more file replicas in the network, and hence a host has more chances to get a file replica from a nearby host, resulting in a lower average hops per access. Therefore, Figure 6.7 is similar to the "inverse" of Figure 6.1. For the same reason, the same similarity can also be seen in other pairs of figures: Figure 6.8 and Figure 6.2, and Figure 6.9 and Figure 6.3.

Figure 6.8 shows the average hops per access as a function of the write percentage in the workload for the four policies. The more write operations there are in the

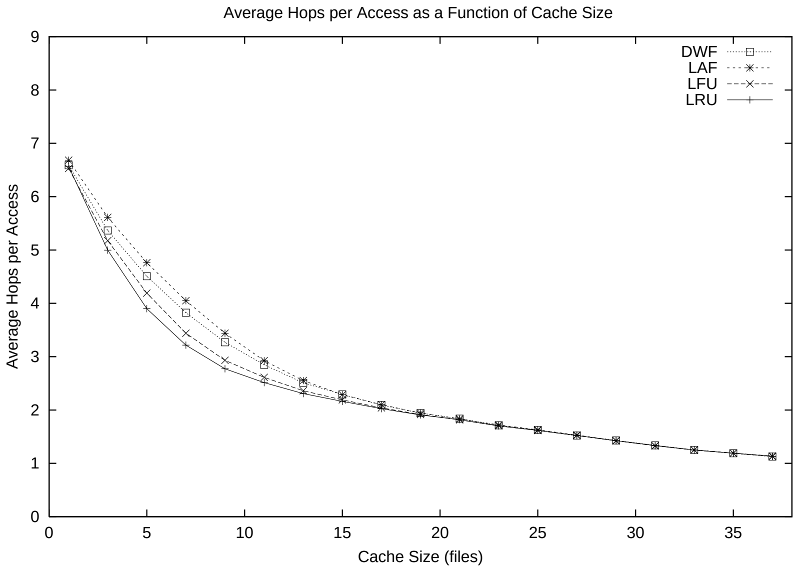

Figure 6.7: Average hops as a function of cache size

workload, the higher the average hops per access becomes. This is because file updates cause the stale file replicas to be invalidated, and result in a small number of file replicas in the network. A host may have to get a requested file replica from a far away host and the average hops per access increases as a result. Among the four policies, LRU seems to perform best.

LAF reacts to write percentage slightly differently than the other policies. Increasing write percentage decreases the average hops per access for LAF at first, but then the average hops per access for LAF increases as the write percentage increases further. LAF replaces the replica with the lowest access frequency. Because of the concentration characteristic of the workload, LAF gradually causes every mobile host to have a similar set of replicas in the cache. This effect causes a host to search more hosts in the network for a request. Thus a successful access is likely to use more hops in LAF than in the other policies, and the chance for an unsuccessful access is high. Therefore, the average hops per access for LAF is high when the write percentage is low. Replica invalidations caused by file updates help hosts to have different sets of replicas by reducing the number of replica replacements performed. Thus the average hops per access decreases for LAF as the write percentage in-

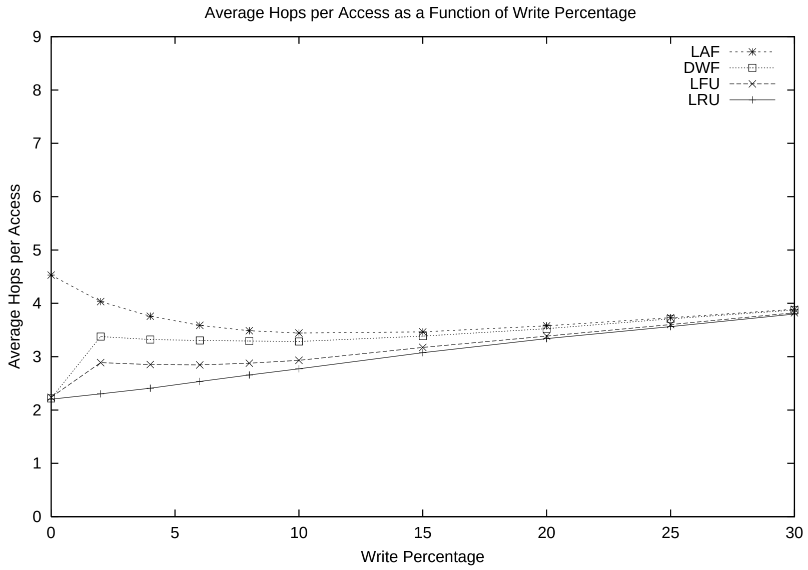

Figure 6.8: Average hops as a function of write percentage

creases. But a large write percentage causes empty spaces in the cache, reduces the number of replica in the network, which in turn, causes more hops for a successful access, and increases the number of unsuccessful accesses. Therefore, the average hops per access increases as the write percentage increases further.

Figure 6.9 shows the effect of radio range on the average hops per access. The radio range affects the average hops per access the same way for the four policies. Increasing radio range significantly decrease the average hops per access when radio range is small. A high radio range means the network gets partitioned rarely, and single-hop communication becomes common in the network. Therefore, the average hops is low when the radio range is high, and further increasing the radio range has little effect on decreasing the average hops per access. Although LRU appears to perform better than the other policies, the differences among the four are very small.

The results show that all three factors, cache size, write percentage in the workload and radio range, affect the average hops per access in a similar way for the four replacement policies considered. LAF is not a good policy because it causes mobile hosts to have similar replica sets in their caches. LRU is slightly better than the

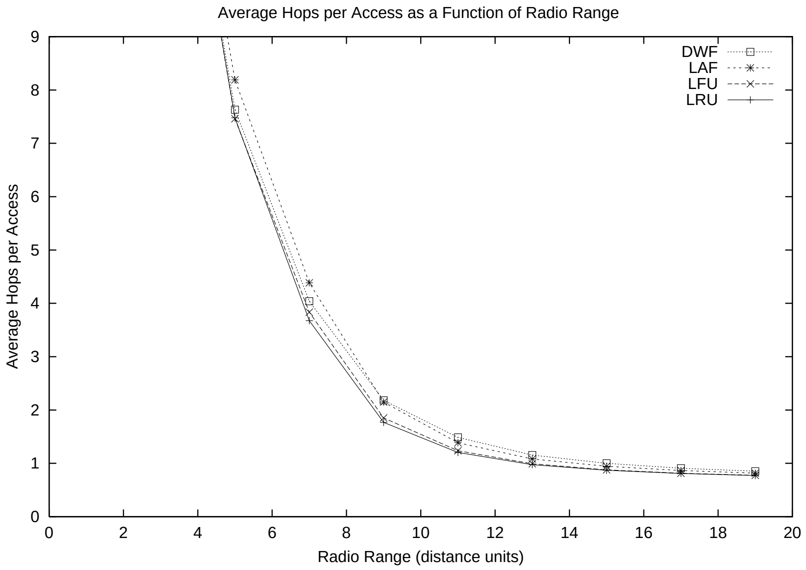

Figure 6.9: Average hops as a function of radio range

others since it can take advantage of the temporal locality and the concentration characteristics in the workload. In general, the differences among the four policies are small.

6.3 Summary

Four replica replacement policies were examined: LRU, LFU, LAF, and DWF. The performance of the four policies, in terms of file accessibility, fresh access and average hops per access, was investigated through simulation experiments. Three factors, cache size, the write percentage in the workload, and the radio range of the mobile hosts, were varied for the experiments.

The results show that the performance of the four policies is quite similar. The performance of LRU appears to be slightly better in most of the situations examined because LRU can capitalize on the concentration characteristic and the temporal locality in the workload. LFU has similar performance to LRU because it also benefits from the concentration and temporal locality characteristics in the workload. LAF replaces the replica with the lowest access frequency and gradually causes mobile hosts to have similar sets of replicas because of the concentration characteristic in the workload. As a result, the number of replicas of different files is reduced in the network, and replica duplication increases. As a result, LAF tends to have poor performance. DWF replaces the replica with the highest write access frequency. The frequently updated replicas are not good candidates for caching because they are likely to be stale. But replacing frequently updated files may result in decreasing the number of the most up-to-date replicas of those files in the network. For this reason, DWF does not perform as well as the others in terms of fresh access.

Although the performance differences among the four policies examined are small, LRU appears to be a better choice overall since it can capitalize on the temporal locality which is a common characteristic in most workloads. It performs well under the situations examined.

Chapter 7

Experiments with Reconciliation Policies

Optimistic data replication allows mobile hosts to perform the file access operations they need, but in the presence of file updates, file replicas in the network could be in an inconsistent state at certain instances. Although a replica management policy has some effect on replica consistency, as has been shown in Section 5.2, replica consistency is maintained mainly by periodic reconciliation processes. Although the mobile hosts in the network have the same reconciliation period, they may initiate reconciliation processes at different times to avoid flooding the network with reconciliation processes.

One of the important decisions a host must make is which host in the network should be chosen as a reconciliation partner. That decision is based on the reconciliation policy used (see Section 3.3 for details). Different reconciliation policies result in different degrees of replica consistency because the reconciliation policy used affects the way file updates propagate. A good reconciliation policy should provide mobile hosts with high access rate to the most up-to-date file replicas, while imposing only moderate additional load on the network.

In this chapter, the performance of two reconciliation policies is investigated. The experimental design is described in Section 7.1. Attention is focused on the reconciliation policies, the performance metrics, and the parameter settings for the experiments. After that, the experimental results are presented and discussed in Section 7.2. The chapter concludes with a brief summary in Section 7.3.

7.1 Experimental Design

A reconciliation policy is used to determine which host should be chosen as a reconciliation partner. Reconciliation processes do not affect replica distribution in the network. Therefore, the reconciliation policy used does not affect file accessibility for different replica management policies.

The results from Chapter 5 show that AF-LRU performs best in terms of file accessibility and fresh access under the Closest Host (CH) reconciliation policy. Therefore, in the experimental design in this chapter, the focus is on examining the performance differences between the two reconciliation policies, Closest Host (CH) and File Owner (FO), under the AF-LRU replica management policy.

Although CH and FO may behave differently under different replica management policies, there is no point in investigating the performance of CH and FO under replica management policies which do not provide high file accessibility.

7.1.1 Policies Investigated

Two reconciliation policies, CH and FO (see Section 3.3), were examined:

- CH (Closest Host) – In CH, a mobile host always finds the closest hosts as reconciliation partners to reconcile its replicas. The disadvantage of this policy is that the closest reconciliation partners may not have the most up-to-date replicas, and so the host loses chances to get the most up-to-date information if one of the most up-to-date replicas is in the partition that the host is currently in. On the other hand, if one of the most up-to-date replicas is in the same partition of the network, a host will have a chance to get the most up-to-date information. In addition, since a host always reconciles with the closest reconciliation partners, CH imposes little network load onto the network.

- FO (File Owner) – In FO, a host always chooses the file owner as its reconciliation partner. The advantage is that the file owner is likely to have the most up-to-date file, since it immediately reconciles with hosts that update replicas of the file, once it hears the updates from the invalidation broadcasts. The disadvantage is that if a host which wants to reconcile its replica is not in the same partition as the file owner, the host loses the chance to get the most up-to-date information even if its neighbours happen to have the most up-to-date replicas. In addition, file owners may be far away from the host so that more hops are needed for reconciliation. Moreover, the owner of a popular file may be very busy reconciling file replicas, and thus can become a performance bottleneck.

7.1.2 Performance Metrics and Factors

Three performance metrics are used in the evaluation of reconciliation policies: fresh access, conflict ratio and reconciliation hops (see Section 3.4.1). Higher fresh access means hosts have fewer chances to be served with stale data. Lower conflict ratio means fewer efforts will be made to resolve conflicts. Small numbers of reconciliation hops mean little load is imposed on the network.

Four factors were examined: cache size, write percentage in the workload, radio range, and reconciliation period (see Section 3.4.2). These were varied over wide ranges to investigate their effects on the performance of a reconciliation policy.

7.1.3 Parameter Settings

The parameter settings for the experiments are shown in Table 7.1. AF-LRU was used as the replica management policy because AF-LRU is the best among the five policies considered. The ranges for other parameters were determined by the preliminary experiments (see Section 5.1.3). Experiments were conducted by varying the values of one parameter at a time while keeping the others fixed at specific values. Table 7.2 shows the values of parameters when they were fixed. The values were chosen to make the comparison between the two reconciliation policies easier.

Table 7.1: Factors and levels for the simulation experiments

Factors	Levels
Replica management policy	AF-LRU
Reconciliation policies	CH, FO
Reconciliation period (time unit)	$5 * 2^n$ where n is from 1 to 11
Cache size	2 to 38 files, in steps of 2 files
Write percent in workload	2%, 4%, 6%, 8%, 10%, 15%, 20%, 25%, 30%
Radio range (distance unit)	1, 3, 5, 7, 9, 11, 13, 15, 17, 19

Table 7.2: Values for the parameters when they were fixed

Parameter	Value
Cache size	10 files
Radio range	8 distance units
Write percent in workload	10%
Reconciliation Period	20 time units

7.2 Experimental Results

The results of the simulation experiments are presented and discussed in the following three sections. Each section addresses one aspect of performance of the two reconciliation policies with respect to different factors.

7.2.1 Fresh Access

Fresh access reflects quality of service. Providing high fresh access is the goal of a reconciliation policy. Figure 7.1 shows fresh access as a function of cache size. When the cache size is smaller than 5 files, FO outperforms CH slightly. A small cache size means the number of replicas of a file is small in the network and the number of the most up-to-date replicas of the file is also small. Therefore, the chance for a host to get the most up-to-date information from a nearby host becomes small. On the other hand, when the file owner hears a invalidation broadcast, it will try to reconcile with the host who just updated the file replica. Thus file owners are likely to have the most up-to-date files. Therefore, FO outperforms CH slightly at small cache sizes. Although the small cache size favors FO slightly, the difference between the two is insignificant. Different situations could possibly cause CH to outperform FO.

When the cache size is large, the number of file replicas in the network becomes large and so the number of the most up-to-date replicas becomes large as well. A host has more chances to get the most up-to-date information from a nearby host (other than the file owners). Therefore, CH outperforms FO.

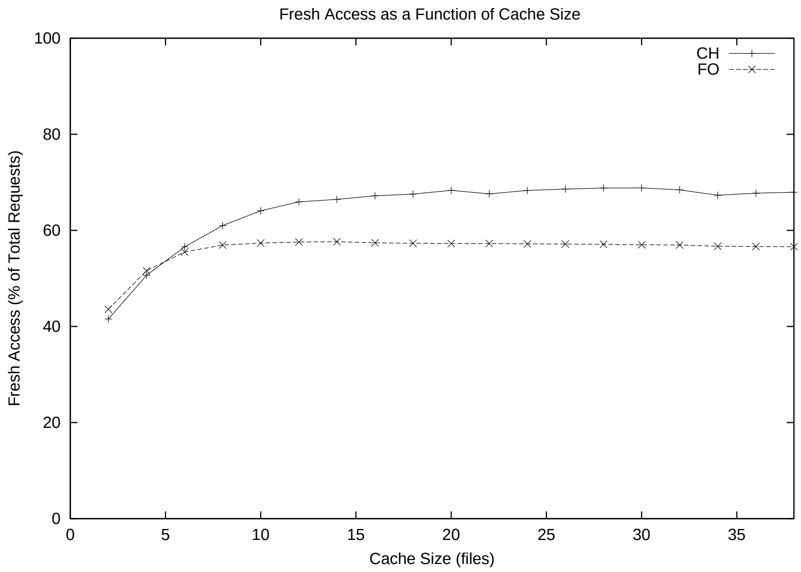

Figure 7.1: Fresh access as a function of cache size

When the cache size exceeds a certain value, further increases do not increase fresh access for either CH or FO. This suggests that there is a “best” cache size for each policy, and the two policies have different “best” cache sizes. Although replica consistency is closely related to the network connectivity, more replicas also cause some difficulties in maintaining replica consistency. Therefore, a tendency of slightly decreasing fresh access can also be seen in Figure 7.1 for both policies.

Figure 7.2 shows the effect of the radio range on the performance of the two reconciliation policies in terms of fresh access. As the radio range increases, fresh access for the two policies increases quickly. CH slightly outperforms FO when the radio range is small. This is because the cache size is 10 files for mobile hosts, which is large enough to make CH outperform FO (as shown in Figure 7.1). As the radio range gets larger, fresh access for both CH and FO significantly increases. This is because the network rarely gets partitioned as the radio range of the mobile hosts gets larger. Because of the improvement in network connectivity, file owners can hear most of the invalidation broadcasts, and a host can always reconcile with file owners. Thus, both FO and CH perform equally well at a large radio range.

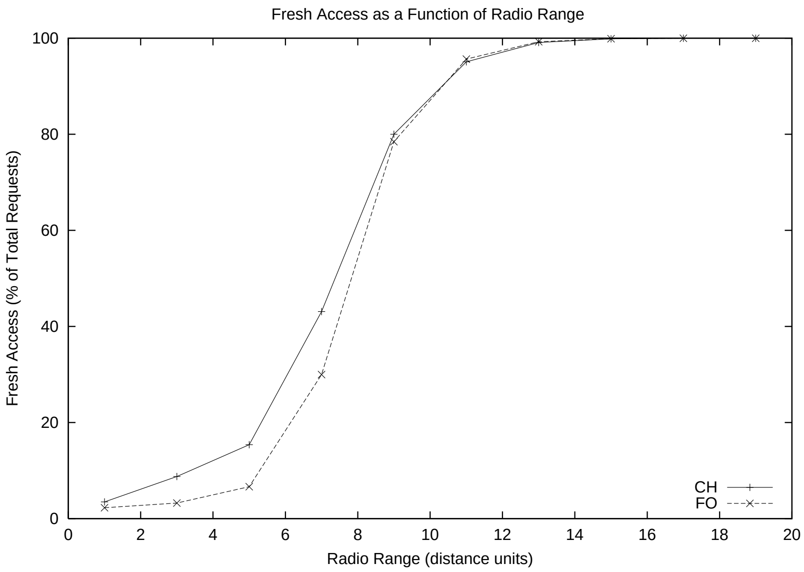

Figure 7.2: Fresh access as a function of radio range

The write percentage in the workload affects fresh access the same way for both CH and FO. Figure 7.3 shows that as the percentage of write operations increases, fresh access quickly decreases for both CH and FO. This is because when there are more write operations in the workload, there are more stale file replicas in the network, and it is more difficult to maintain replica consistency. CH slightly outperforms FO.

As expected, a short reconciliation period results in high fresh access for both CH and FO. Figure 7.4 shows fresh access as a function of reconciliation period. Increasing the reconciliation period decreases fresh access for the two policies. FO is slightly more resistant to the change of reconciliation period than CH. Therefore, when the reconciliation period exceeds 200 time units, FO slightly outperforms CH.

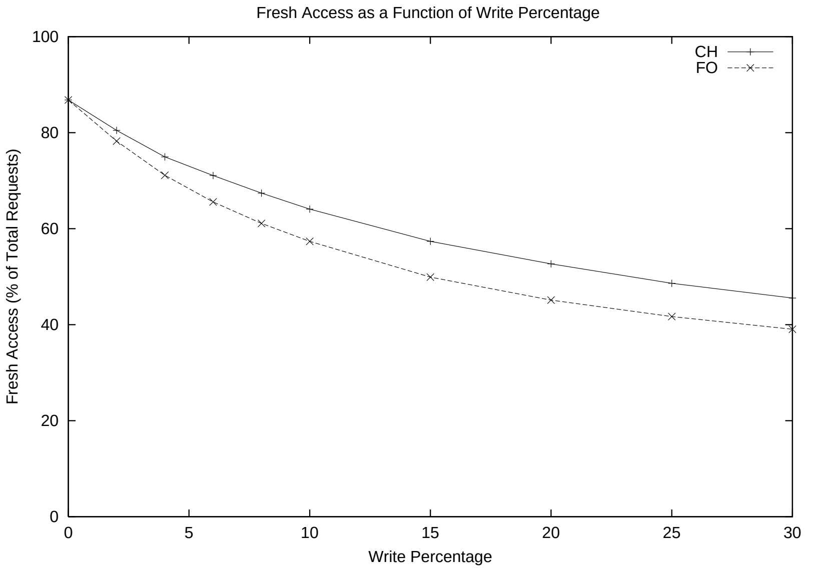

Figure 7.3: Fresh access as a function of write percentage

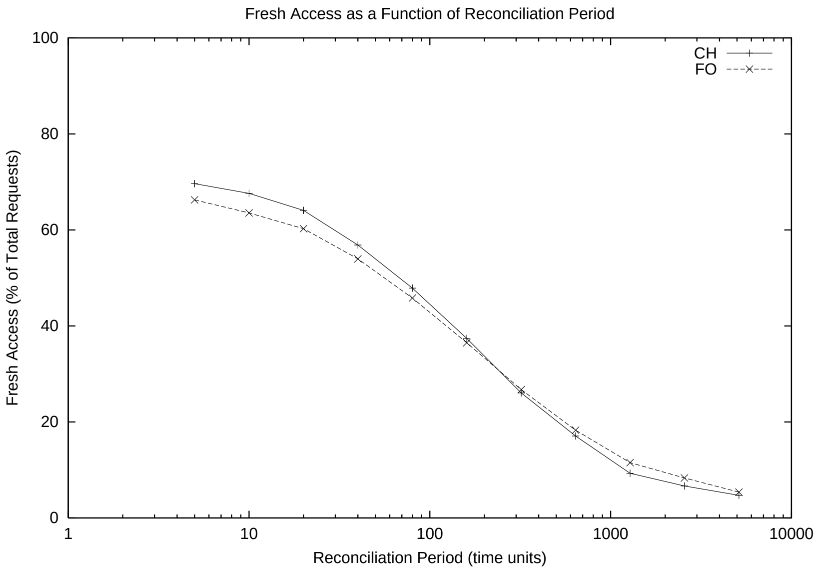

Figure 7.4: Fresh access as a function of reconciliation period

For example, in a network partition, if a host which is not the owner of a file happens to have the most up-to-date replica of the file, the other hosts in the same partition can not get the most up-to-date information using FO, and shortening the reconciliation period will not help because of the network partitions. But all other hosts may get the most up-to-date information using CH provided that the reconciliation period is short enough. Therefore, when the reconciliation period is short, CH outperforms FO. When the reconciliation period is long, FO outperforms CH slightly because file owners are likely to have the most up-to-date information.

The results show that different situations result in different performance for CH and FO in terms of fresh access. But in general, CH outperforms FO in the situations where mobile hosts have moderate cache size, the radio range of mobile hosts is not very large, and the reconciliation period is short.

7.2.2 Conflict Ratio

The conflict ratio is defined as the ratio of the total number of conflicting write accesses to the total number of write access requests in the workload. A higher conflict ratio indicates that more effort is required to resolve the conflicts. Therefore, a small conflict ratio is always desirable.

The conflict ratio correlates with fresh access, which was discussed in Section 7.2.1. Higher fresh access means a higher level of replica consistency, and a lower conflict ratio. Figure 7.5 shows the conflict ratio as a function of cache size. Because of the correlation between fresh access and the conflict ratio, Figure 7.5 is similar to the "inverse" of Figure 7.1.

When the cache size is smaller than 6 files, FO gives a slightly lower conflict ratio than CH does because the number of the most up-to-date replicas is small, and hence the chance for a host to get the most up-to-date information from a nearby host (other than the file owner) is small. A large cache size benefits CH because the large number of the most up-to-date replicas makes a host have more chances to get the most up-to-date information from a nearby host. Thus, CH gives a lower conflict ratio than FO does when the cache size is larger than 6 files. Nonetheless, increasing the cache size results in increasing the conflict ratio for both CH and FO because a larger cache results in a larger number of replicas in the network, which in turn, makes it more difficult to maintain replica consistency.

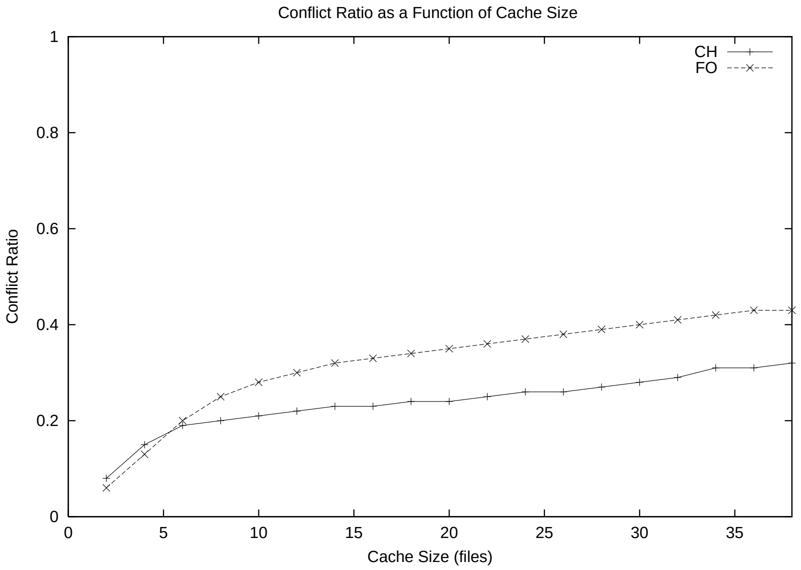

Figure 7.5: Conflict ratio as a function of cache size

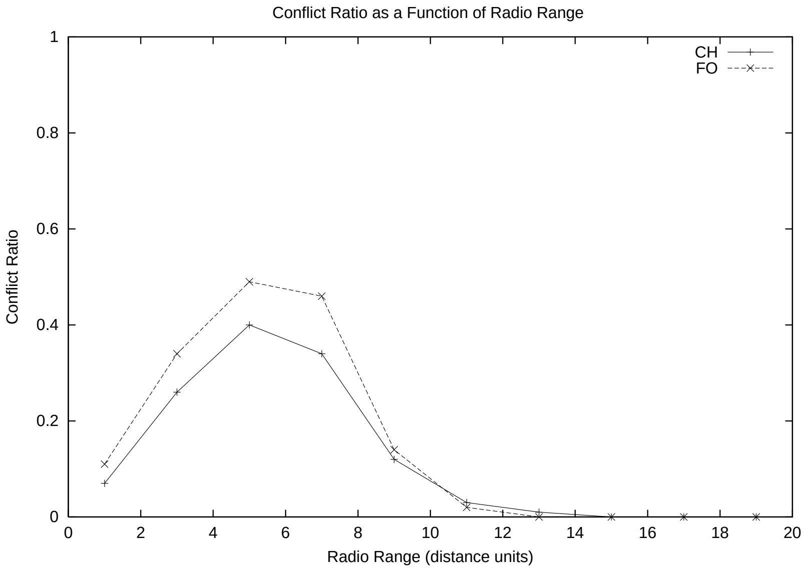

Figure 7.6: Conflict ratio as a function of radio range

Figure 7.6 shows the effect of the radio range on the conflict ratio for the two policies. As the radio range increases, the conflict ratio for both policies increases at first, then decreases once the radio range exceeds a certain value. A low radio range causes frequently network partitioning and a small number of successful write access requests as a result, which in turn, results in a lower conflict ratio. As the radio range increases, the network connectivity improves, therefore the number of successful write access requests increases. This also results in a large number of stale replicas, making it more difficult to maintain replica consistency, and increasing the conflict ratio. As the radio range exceeds a certain value, the conflict ratios for both CH and FO decrease as the radio range increases further. This is because a large radio range makes the network partition infrequently, and so stale replicas can be easily invalidated. In the extreme, when the radio range is large enough, the conflict ratio for both policies becomes zero.

Figure 7.7 shows conflict ratio as a function of reconciliation period for both policies. Not surprisingly, increasing the reconciliation period results in increasing the conflict ratio for both policies. Again, because of the correlation between fresh access and the conflict ratio, Figure 7.7 is similar to the "inverse" of Figure 7.4. A smaller reconciliation period makes CH perform a little better than FO, and both policies perform equally poorly at a large reconciliation period.

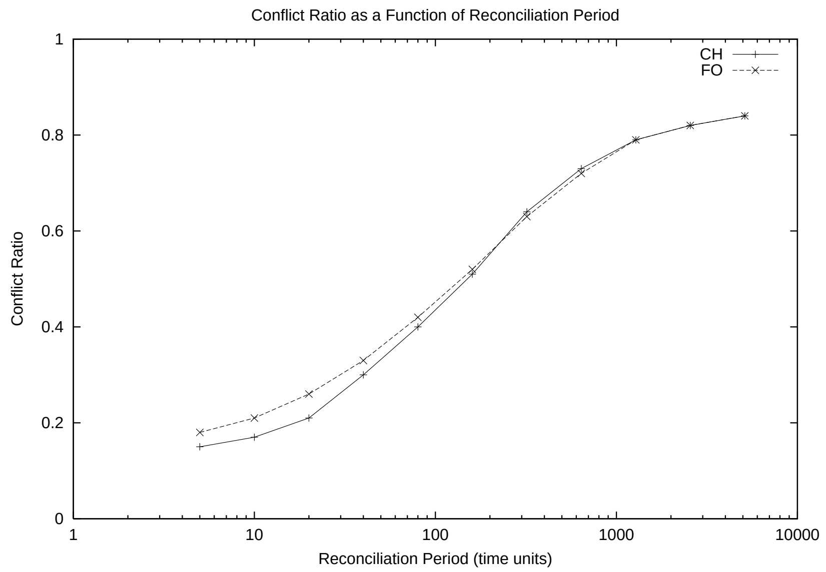

Figure 7.7: Conflict ratio as a function of reconciliation period

Figure 7.8 shows the conflict ratio as a function of the write percentage in the workload. Increasing the write percentage in the workload makes it difficult for both

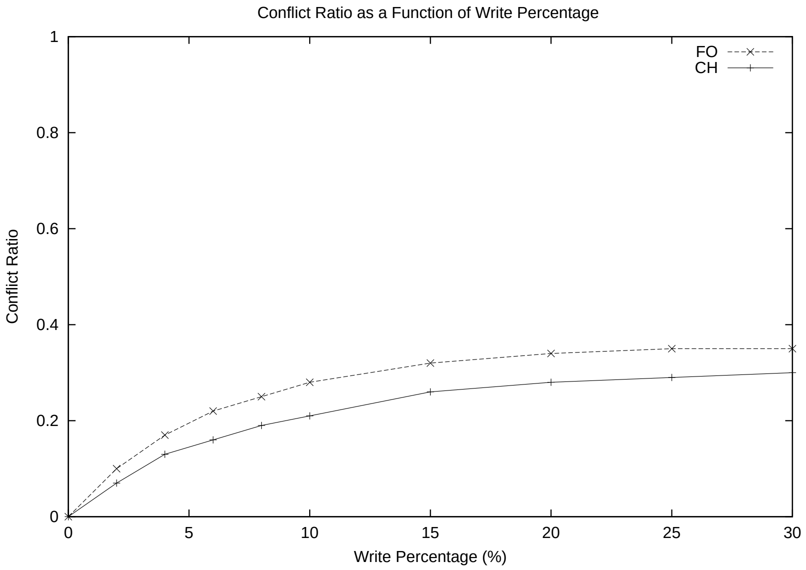

Figure 7.8: Conflict ratio as a function of write percentage

CH and FO to maintain replica consistency. Thus the conflict ratio increases as the write percentage increases. In the range of write percentages examined, the conflict ratio for CH is lower than that for FO. But the difference between them is small.

The results show that the four factors affect the conflict ratio in the same way as they affect fresh access for both FO and CH. When the cache size is large and the reconciliation period is short, CH outperforms FO in terms of conflict ratio.

7.2.3 Reconciliation Hops

The number of reconciliation hops reflects the network load caused by a reconciliation policy. It is defined as the total hop count of data transmission for reconciling file replicas during the whole simulation time. For CH, the reconciliation hops is the total hop count for replica reconciling between a host and its available closest host. For FO, the reconciliation hops is the total hop count for replica reconciling between a host and the file owner. Since a wireless network has less bandwidth than a wired network, reducing network usage is an important issue in a mobile ad hoc network environment.

Figure 7.9 shows the reconciliation hops as a function of cache size. Not surprisingly, increasing the cache size results in increasing the reconciliation hops for both CH and FO. This is because the large cache size allows a mobile host to hold many replicas in the cache, resulting in a large number of replicas in the network. Therefore, a large number of reconciliation processes are performed, causing a large number of reconciliation hops. In the range of cache sizes considered, FO uses fewer reconciliation hops than CH does because FO requires a host to reconcile a replica with the file owner, which may not be available at the time because of the network partitions. Thus FO performs fewer reconciliation processes than CH does, which allows a host to reconcile a file replica with another replica in the same network partition.

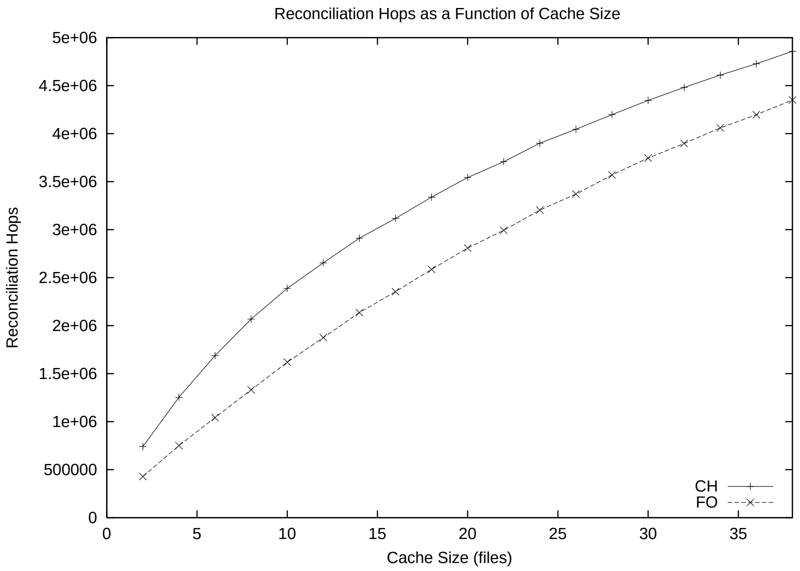

Figure 7.9: Reconciliation hops as a function of cache size

Figure 7.10 shows the reconciliation hops as a function of the write percentage in the workload. As the write percentage increases, the reconciliation hops decreases in the same way for both CH and FO. A write access will cause a invalidation broadcast to invalidate the stale replicas. Thus frequent write accessing results in a small number of replicas in the network, which in turn causes a small number of reconciliation processes to be performed. Therefore, the total number of reconciliation hops decreases as the write percentage in the workload increases. Again, in the range of write percentages examined, FO uses fewer reconciliation hops than CH does for the same reason described above.

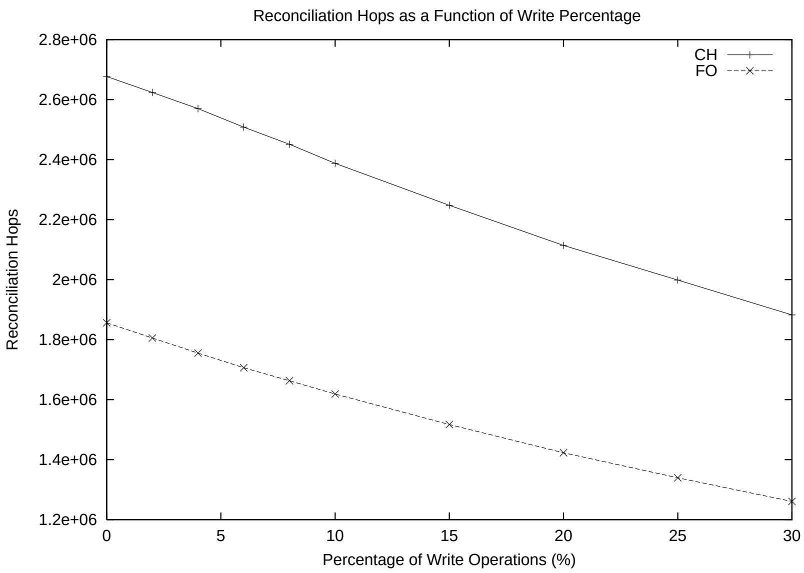

Figure 7.10: Reconciliation hops as a function of write percentage

The effect of the radio range of the mobile hosts on the reconciliation hops is shown in Figure 7.11. When the radio range is small, the network gets partitioned frequently. The chance for a host to be able to reconcile replicas with the file owners is small. Thus, FO performs fewer reconciliation processes and uses fewer reconciliation hops than CH does. As the radio range increases, the network rarely gets partitioned so that the chance for a host to be able to reconcile replicas with the file owners increases. Since the file owners may be far away from the host who want to reconcile its replicas, the communication paths between the host and the file owners are mostly multi-hop paths. Therefore, when the radio range is large enough, FO uses more reconciliation hops than CH does. Although further increasing the radio range decreases the reconciliation hops for FO, FO still uses more reconciliation hops than CH does because CH always tries to reconcile with the closest hosts.

As expected, decreasing reconciliation period will dramatically increase reconciliation hops for both FO and CH policies for the obvious reason that shorter

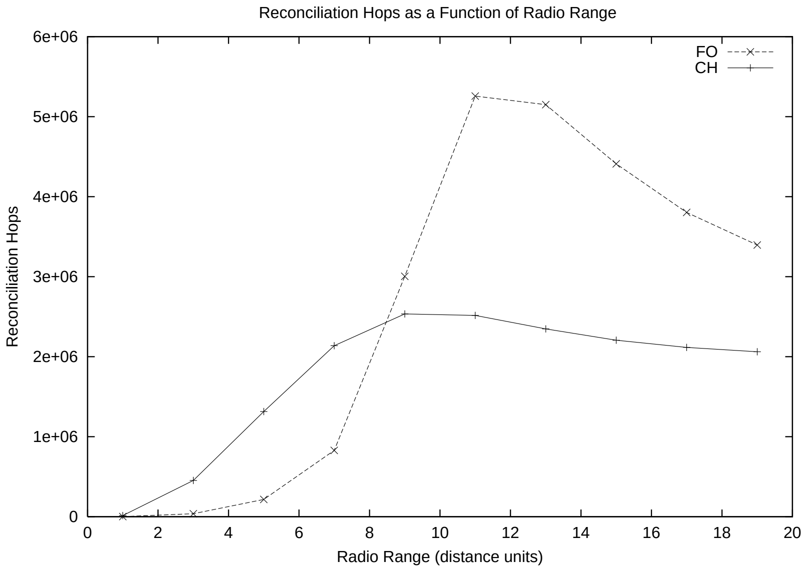

Figure 7.11: Reconciliation hops as a function of radio range

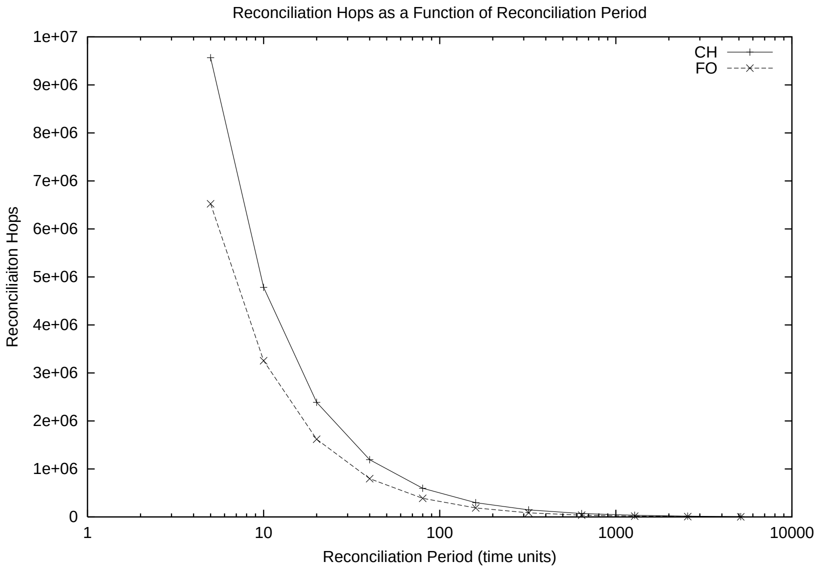

Figure 7.12: Reconciliation hops as a function of reconciliation period

reconciliation periods make hosts perform more reconciliations. Figure 7.12 shows the total number of reconciliation hops used by FO and CH policies under different reconciliation periods. Although a short reconciliation period results in a high reconciliation hops, it also results in a high level of replica consistency as shown in

Figure 7.4. Hence, there is a trade-off between the reconciliation period and the level of replica consistency. To maintain a higher level of replica consistency, a shorter reconciliation period is needed.

In general, CH uses more reconciliation hops than FO does under most situations considered when the radio range is small.

7.3 Summary

Reconciliation is an important issue in optimistic replication schemes. In this Chapter, two pairwise reconciliation policies, FO and CH, were investigated under the AF-LRU replica management policy. The effects of four important factors were examined: the cache size, the write percentage in the workload, the radio range of the mobile hosts and the reconciliation period.

The reconciliation period has a significant impact on performance for both CH and FO. A shorter reconciliation period provides a higher level of replica consistency in terms of fresh access, but it also imposes more load on the network measured in reconciliation hops. Therefore, a balance should be made between maintaining replica consistency and reducing network load.

The results show that different situations result in different conclusions in regard to performance comparisons between FO and CH. CH performances moderately better than FO in terms of fresh access and conflict ratio under most of the situations considered. But CH imposes more network load in terms of the reconciliation hops. Therefore, if network usage is the major concern, but the quality of service (i.e., accessing stale replicas is acceptable) is less important, FO could be a better choice than CH. If the quality of the service is the main concern, CH is a better choice.

Chapter 8

Summary, Conclusions and Future Work

This thesis address issues related to file sharing, include read and write, among mobile hosts on a mobile ad hoc network. The performance of five replica management policies, four replica replacement policies, and two reconciliation policies, was studied through simulation experiments. This chapter summarizes the work that was done and presents some possible directions for future work in this area.

8.1 Summary of the Thesis

The thesis consists of three parts. Chapter 1 and Chapter 2 form the first part, which provides background information about ad hoc networks, optimistic replication, and previous research related to the topic.

The second part is formed by Chapter 3 and Chapter 4. In this part, the mobile ad hoc network system used in this study is described, and policies are proposed to support file sharing among mobile hosts by using optimistic replication combined with a peer-to-peer model. The system operations and the interactions among mobile hosts are defined. The performance metrics and the important factors affecting performance are identified. Finally, the simulation model and the simulation environment are described.

The performance evaluation of the proposed policies is in the third part of the thesis, which consists of Chapter 5, Chapter 6, and Chapter 7. In this part, the results of the simulation experiments are presented and discussed.

In Chapter 5, the performance of five replica management policies was evaluated. In most situations considered, three on-demand policies, AF-LRU, FS-LRU, LRU-Only outperform two periodic frequency-based policies, SAF and DAFN, in terms of file accessibility, fresh access, response time and network load. Among the three on-demand policies, AF-LRU gives the best performance.

In Chapter 6, four replica replacement policies, LRU, LFU, LAF, and DAFN were examined as a part of replica management. The results show that the performance of the four policies is quite similar. Since LRU can capitalize on the temporal locality in the workload, it slightly outperforms the other policies.

In Chapter 7, the performance of two reconciliation policies used to maintain replica consistency using pairwise reconciliation processes was evaluated. The results show CH is better than FO in terms of fresh access in most situations, but the difference between them is small (around 10%). Since CH imposes more network load measured in the reconciliation hops than FO does, FO may be preferred if network usage is a concern and the particular applications can tolerate slightly out-of-date files.

8.2 Conclusions

Optimistic replication combining with peer-to-peer model can be used to support file sharing, including read and write, on mobile ad hoc networks. Peer-to-peer optimistic replication has been used to support mobile users in systems such as Ficus [27] and Bayou [57], but it has not been used on a mobile ad hoc network. In previous work by Hara [29], replication is used to support data sharing, but write sharing is not included. To the best of our knowledge, this study marks the first attempt to investigate the possibility of supporting full file sharing, including read and write, on mobile ad hoc networks using peer-to-peer optimistic replication.

To support file sharing using replication on a mobile ad hoc network, it is very important to share replicas of different files among mobile hosts. In SAF, mobile hosts allocate file replicas according to the access frequencies to files, resulting in many replica duplications among mobile hosts because of the concentration characteristic in the workload. In DAFN, extra effort is devoted to removing duplicate replicas resulting in a larger number of replicas of different files among neighbours, and therefore results in a better performance than SAF. On-demand allocation, in which a mobile host allocates file replicas when it successfully accesses the files, makes mobile hosts instinctively share replicas of different files provided that mobile hosts access different files at different times, which is the case in most workloads. For this reason, the three on-demand policies generally outperform the two periodic frequency-based policies.

With respect to the network load imposed, the on-demand policies again outperform periodic frequency-based policies. The obvious reason is that the on-demand policies allow hosts to replicate files as hosts successfully access the files, and thus reduce the hops needed for transferring replicas from other hosts for replica allocation.

One drawback associated with on-demand approaches is that the cache space may not be effectively utilized. When the cache size is large, the write percentage in the workload is high, and the network partitions frequently (i.e. a small radio range causes frequent network partitioning), it takes a long time to fill up the cache space. To overcome this problem, AF-LRU allows a host to check its cache periodically. If there is enough empty space in the cache, the host allocates some replicas according to the access frequencies to fill up the empty cache space. This extra effort makes AF-LRU perform best among the five policies considered at the cost of using extra hops for transferring replicas from other hosts.

Different replica replacement policies have different impacts on the performance of replica management. But the difference among them is small. When the cache size is small, the choice of replicas is limited; when the cache size is large, replica replacement occurs rarely. Thus, only when the cache is of moderate size, the performance differences are larger (around 10%). Because LRU can capitalize on the temporal locality in the workload, it gives slightly better performance and appears to be a better choice.

Of the two reconciliation policies used to maintain consistency, CH is better than FO in terms of fresh access in most situations examined. Since CH chooses the closest host as the reconciliation partner, it is more resilient to network partitioning than FO which selects the file owner as the reconciliation partner. For example, if a host which is not the owner happens to have the most up-to-date replica in a network partition, the other hosts in the same partition can get the most up-to-date information using CH provided the reconciliation period is small enough. But they cannot get the most up-to-date information using FO. Since FO always selects the file owner, the owner of a popular file could become a performance bottleneck. Therefore, CH is a better choice than FO in general.

The results from this study indicate that it is possible to use peer-to-peer optimistic replication technique to support file sharing on a mobile ad hoc network.

8.3 Future Work

There are many issues related to file sharing support on mobile ad hoc networks and it is impossible to address them all in one research project. This thesis addresses only some of them, and under restricted situations. The following are some possible future research directions as a continuation of this study:

- Characteristics of workload – In this study, it was assumed that files are independent of each other. Correlations among files would definitely impact system performance. In addition, file updates may also have correlations among each other because of the correlations among the files. Therefore, a replica management could take into account of correlations among files to improve performance. For example, neighbouring hosts should allocate replicas of correlated files in order to help each other.

- Mobility model – In this study, a simple group mobility model (see Section 3.1.1) was used to model a conference situation. It is unclear how a more complicated mobility pattern would affect performance. For example, some hosts may move slowly in a particular area but quickly in another area in a disaster relief situation. This may cause different partition patterns in a mobile ad hoc network, and thus affect performance.

- Reconciliation policy – If correlations among files are considered, the reconciliation policy may need to take these correlations into account. In some situations, a file is updated and some related files must be updated as well. It is possible to utilize the correlations among the files to find the reconciliation partners to provide higher level of replica consistency than CH or FO does. For example, if the replicas of correlated files are allocated among neighbouring hosts because of the correlations, one of the neighbouring hosts is chosen as a reconciliation partner, the others should be chosen as well.

- Detailed simulation model – Because this study is a first step in an investigation of file sharing support among mobile hosts on a mobile ad hoc network, the details of the ad hoc network were not simulated. Those details may affect the performance of proposed policies. For example, it was assumed that files can be transferred instantly between two hosts, and so file size has no impact on performance. If more details of the network are simulated, file transfer between hosts will take some time, and file size may have some impact on the performance of proposed policies. Therefore, a more detailed simulation model could be used to further investigate proposed policies.

References

[1] M. Ahamad and R. Kordale. Scalable consistency protocols for distributed services. *IEEE Transactions on Parallel and Distributed Systems*, 10(9):888–903, Sept. 1999.

[2] Y. Amir and C. Tutu. From total order to database replication. In *Proc. of the 22nd IEEE International Conference on Distributed Computing Systems*, pages 1576–1586, Vienna, Austria, July 2002.

[3] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In *Proc. of the 15th ACM Symposium on Operating Systems Principles*, pages 109–126, Copper Mountain Resort, CO, Dec. 1995.

[4] M. Arlitt and C. L. Williamson. Web server workload characterization: The search for invariants. In *Proc. of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 126–137, Philadelphia, PA, April 1996.

[5] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proc. of the 1998 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 151–160, Madison, WI, June 1998.

[6] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publishing Company, 1987.

[7] J. Binkley. Authenticated ad hoc routing at the link layer for mobile systems. Technical report 96–3, Department of Computer Science, Portland State University, Portland, OR, 1996.

[8] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. of ACM Mobicom 98*, pages 85–97, Dallas, TX, Oct. 1998.

[9] J. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proc. of IEEE INFOCOM 1999*, pages 275–283, New York, NY, April 1999.

[10] T. Camp, J. Boleng, and V. Davies. Mobility models for ad hoc network research. *Wireless Communications & Mobile Computing: Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[11] U. Cetintemel and P. J. Keleher. Light-weight currency management mechanisms in deno. In *Proc. of the 10th IEEE Workshop on Research Issues in Data Engineering*, pages 10–21, San Diego, CA, Feb. 2000.

[12] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[13] P. Charles and E. Royer. Ad hoc on-demand distance vector routing. In *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, Feb. 1999.

[14] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46–66, 2001.

[15] D. Stott Parker Jr., G. Popek, G. R. A. Stoughton, B. Walker, E. Walton, J. . Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Transactions on Software Engineering*, 9(3):240–247, May 1983.

[16] S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):343–369, Sept. 1985.

[17] M. Ebling and M. Satyanarayanan. Synrgen: An extensible file reference generator. In *Proc. of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 108–117, Nashville, TN, May 1994.

[18] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2):374–382, April 1985.

[19] G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, 27(4):38–47, April 1994.

[20] M. Frodigh. *Wireless ad hoc networking – The art of networking without a network*. Available at www.ericsson.com/review/2000_04/article124.shtml, 2000.

[21] K. Froese. *File Cache Management for Mobile Computing*. M.Sc. thesis, Dept. of Computer Science, University of Saskatchewan, Saskatoon, SK, 1997.

[22] D. Gifford. Weighted voting for replicated data. In *Proc. of the 7th ACM Symposium on Operating Systems Principles*, pages 150–162, Dec. 1979.

[23] Gnutella. http://gnutella.wego.com.

[24] S. Goel, M. Singh, D. Xu, and B. Li. Efficient peer-to-peer data dissemination in mobile ad-hoc networks. In *Proc. of International Workshop on Ad Hoc Networking*, pages 152–158, Vancouver, BC, Aug. 2002.

[25] R. Golding. Modeling replica divergence in a weak-consistency protocol for global-scale distributed databases. Technical report, Computer and Information Science, University of California, Santa Cruz, CA, 1993.

[26] B. Gronvall, A. Westerlund, and S. Pink. The design of a multicast-based distributed file system. In *Proc. of the 3rd ACM Symposium on Operating Systems Design and Implementation*, pages 251–264, New Orleans, LA, Feb. 1999.

[27] R. Guy, J. Heidemann, W. Mak, T. Page, G. Popek, and D. Rothmeier. Implementation of the ficus replicated file system. In *Proc. of Summer USENIX Conference*, pages 63–71, Anaheim, CA, June 1990.

[28] R. Guy, P. Reicher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile data access through optimistic peer-to-peer replication. In *Proc. of the 17th International Conference on Conceptual Modeling*, pages 254–265, Singapore, Nov. 1998.

[29] T. Hara. Effective replica allocation in ad hoc networks for improving data accessibility. In *Proc. of IEEE Infocom 2001*, pages 1576–1586, Anchorage, Ak, April 2001.

[30] J. Heidemann, T. Page, R. Guy, and G. Popek. Primarily disconnected operation: Experiences with ficus. Technical report, Department of Computer Science, University of California, Los Angeles, CA, 1992.

[31] M. Herlihy. A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems*, 4(1):32–53, Feb. 1986.

[32] J. H. Howard. An overview of the andrew file system. In *Proc. of Winter USENIX Conference*, pages 134–145, Dallas, TX, Feb. 1988.

[33] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353, pages 153–181. Kluwer Academic Publishers, 1996.

[34] G. Karumanchi, S. Muralidharan, and R. Prakash. Information dissemination in partitionable mobile ad hoc networks. In *Proc. of the 18th IEEE Symposium on Reliable Distributed Systems*, pages 4–13, Lausanne, Switzerland, Oct. 1999.

[35] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *Proc. of the 18th Internationnal Conference on Distributed Computing Systems*, pages 1576–1586, Amsterdam, The Netherlands, May 1998.

[36] T. Ken and G. Mario. Reliable on-demand multicast routing with congestion control in wireless ad hoc networks. In *Proc. of SPIE 2001*, pages 10–21, Denver, CO, Aug. 2001.

[37] J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10(1):3–23, Jan. 1992.

[38] G. Kuenning, R. Bagrodia, R. Guy, G. Popek, and P. Reiher. Measuring the quality of service of optimistic replication. In *ECOOP Workshops on Mobility and Replication*, pages 319–320, Brussels, Belgium, July 1998.

[39] P. Kumar and M. Satyanarayanana. Flexible and safe resolution of file conflicts. In *Proc. of Winter USENIX Conference*, pages 95–106, New Orleans, LA, Jan. 1995.

[40] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communication of the ACM*, 21(7):558–565, July 1978.

[41] H. Luo and S. Lu. Ubiquitous and robust authentication services for ad hoc wireless networks. Technical report, Department of Computer Science, University of California, Los Angeles, CA, 2000.

[42] T. Minoura and G. Wiederhold. Resilient extended true-copy token scheme for a distributed database system. *IEEE Transactions on Software Engineering (TSE)*, 8(3):173–189, May 1982.

[43] E. Pacitti and E. Simon. Update propagation strategies to improve freshness in lazy master replicated databases. *VLDB Journal: Very Large Data Bases*, 8(3–4):305–318, 2000.

[44] T. Page, R. Guy, J. Heidemann, D. Ratner, A. Goel, G. Kuenning, and G. Popek. Perspectives on optimistically replicated peer-to-peer filing. *Software–Practice and Experience*, 28(2):155–180, Feb. 1998.

[45] M. Papadopuli and H. Schulzrinne. Design and implementation of a peer-to-peer data dissemination and prefetching tool for mobile users. In *Proc. of the First NY Metro Area Networking Workshop*, pages 10–21, IBM TJ Watson Research Center, Hawthorne, NY, March 2001.

[46] D. Parker, Jr., G. Rudisin, A. Sroughton, B. Walker, E. Walton, J. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed system. *IEEE Transactions on Software Engineering*, 9(3):240–247, May 1983.

[47] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. of the ACM SIGCOM'94 Conference on Communications Architectures, Protocols and Application*, pages 234–244, London, England, Aug. 1994.

[48] K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and A. Demers. Flexible update propagation for weakly consistent replication. In *Proc. of the 16th ACM Symposium on Operating Systems Principles*, pages 288–302, Saint-Malo, France, Oct. 1997.

[49] D. Ratner, P. Reiher, and G. Popek. Dynamic version vector maintenance. Technical report, Department of Computer Science, University of California, Los Angeles, CA, 1997.

[50] D. H. Ratner. Selective replication: Fine-grain control of replicated file. Technical report, Department of Computer Science, University of California, Los Angeles, CA, 1995.

[51] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, and G. Popek. Resolving file conflicts in the ficus file system. In *Proc. of Summer USENIX Conference*, pages 183–195, Boston, MA, June 1994.

[52] E. Royer. and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communication*, vol. 6(2):46–55, April 1999.

[53] Y. Saito. Optimistic replication algorithms. Technical report, Department of Computer Science, University of Washington, Seattle, WA, 2000.

[54] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the sun network file system. In *Proc. of Summer USENIX Conference*, pages 119–130, Protland, OR, June 1985.

[55] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.

[56] L. Sung-Ju, S. William, H. Julian, G.Mario, and B. Rajive. A performance comparison study of ad hoc wireless multicast protocols. In *Proc. of IEEE INFOCOM 2000*, pages 565–574, Tel Aviv, Israel, March 2000.

[57] D. Terry, M. Theimer, K. Peterson, A. Demers, M. Spreitzer, and C. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proc. of the 15th ACM Symposium on Operating Systems Principles*, pages 172–183, Cooper Mountain, CO., Dec. 1995.

[58] R. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.

[59] C. Williamson and R. Bunt. Characterizing short-term file referencing behaviour. In *Proc. of the 5th IEEE Annual Phoenix Conference on Computers and Communications*, pages 651–660, Phoenix, AZ, March 1986.

[60] D. Willick, D. Eager, and R. Bunt. Disk cache replacement policies for network fileservers. In *Proc. of the 13rd International Conference on Distributed Computing Systems*, pages 2–11, Pittsburgh, PA, May 1993.

[61] L. Zhou and Z. Haas. Securing ad hoc networks. *IEEE Networks*, 13(6):24–30, Nov. 1999.