A Performance Evaluation of Dynamic Replication In Mobile Ad-hoc Networks

by

Wenbo Jiang

A thesis submitted to the School of Computing

in conformity with the requirements for

the degree of Master of Science

Queen's University

Kingston, Ontario Canada

September, 2006

Your file Votre référence
ISBN: 978-0-494-18746-3
Our file Notre référence
ISBN: 978-0-494-18746-3

ABSTRACT

Much research has been done in mobile ad-hoc networks (MANETs), which are multi-hop mobile wireless networks. Providing data services in MANETs, however, is not as well-studied. The advantages of MANETs are limited without data services. Data replication provides an effective way to improve data access performance and enhance data availability in MANETs. Data replication, however, faces significant challenges in MANETs because of their lack of an infrastructure and their dynamic network topologies. Furthermore, node mobility causes frequent network partitioning and maintaining replica consistency becomes extremely difficult in MANETs. It is imperative that data is distributed intelligently to achieve better performance and availability in such a dynamic environment.

In this thesis, we study the data access performance, data availability, data consistency and energy consumption of replication algorithms in MANETs. We conduct simulations on two dynamic replication algorithms, namely FDRM and ARAM, and a static algorithm, namely SDRM, and MFDRM, which is our extension of FDRM and ARAM. Simulation results demonstrate that current replication algorithms are not adequate for MANETs. Based on our simulation results, we make several suggestions on the design of replication systems in MANETs

Keywords: Data Replication, Mobile ad-hoc networks (MANETs), Read-One-Write-All (ROWA), Network Simulator Version 2 (NS-2) and Ad-hoc On-Demand Distance Vector Routing (AODV)

Acknowledgements

The past three years at Queen's have been the most challenging and exciting in my life and I have learned so much. I would like to express my sincere thanks to my supervisors Dr. Hossam Hassanein and Dr. Patrick Martin for their guidance, advice and support during my study at Queen's University. They are excellent advisors and gave me tremendous help in my research and thesis writing. I truly believe this thesis could not have been finished without their help.

Thanks to everyone in the Telecommunications Research Lab and the Database Research Lab for their friendship, advice, and encouragement during my study. Special thanks are given to the School of Computing and Information Science at Queen's University for giving me the opportunity to pursue my graduate studies. I also thank CITO for providing financial support.

Finally, I would like to thank my wife Sally Han for her encouragement, support and sacrifices during my study. Special thanks are also given to my mother-in-law Yunnan Wang for taking care of my son Christopher Jiang who was born during my studies.

Table of Contents

LIST OF FIGURES

List of Tables

Glossary of Acronyms

MANETs	Mobile ad-hoc networks
AODV	Ad-hoc On - Demand Distance Vector
DSR	Dynamic Source Routing
DSDV	Destination - Sequenced Distance Vector
BS	Base Station
DFSA	Deterministic Finite State Automaton
IP	Internet Protocol
LAN	Local Area Networks
MAC	Medium Access Control
NS-2	Network Simulator 2
QC	Quorum Consensus
ROWA	Read-One-Write-All
CBR	Constant Bit Rate
PDA	Personal Data Assistant
ORMDP	On-Demand Multicast Routing Protocol
TORA	Temporally-Ordered Routing Algorithm
STRA	Source-Tree Routing in Wireless Networks
TBRPF	Topology Dissemination Based on Reverse-Path Forwarding
ARAM	Dynamic Adaptive Replica Allocation Algorithm in MANETs
FDRM	Dynamic Distributed Replica Management Mechanism Based On Accessing Frequency Detecting
DHTR	Distributed Hash Table Replication

Chapter 1

Introduction

1.1 Motivation

Recent advances in radio communication and computer technologies have led to an increasing interest in mobile ad-hoc networks (MANETs). MANETs are, however, far from maturity and there are tremendous difficulties in making MANETs into real working systems. Most previous research work in MANETs focused on routing protocols [31], which allow a mobile node to efficiently discover routes between communication peers. Although routing is an important and fundamental issue in MANETs, we believe providing data services is another important goal in MANETs. Unfortunately little research into providing data services in MANETs has been done. Previous research work [9] demonstrates that data replication is a feasible solution to provide efficient and reliable data service in distributed systems. Data replication, however, faces significant challenges in MANETs because of their lack of an infrastructure and their dynamic network topologies.

There are basically two different ways to configure mobile wireless networks [2]. They are infrastructure-based mobile wireless networks (Figure 1.1) and ad-hoc based mobile wireless networks (Figure 1. 2). Infrastructure-based mobile networks are the most prominent in wireless LAN and global wireless networks. An infrastructure-based mobile network uses fixed network access points, also known as base stations, with which mobile nodes interact for communication. For example, a base station forwards messages that are sent or received by mobile nodes. One limitation of infrastructure-

based mobile wireless networks is that mobile nodes must be in the communication range of base stations.

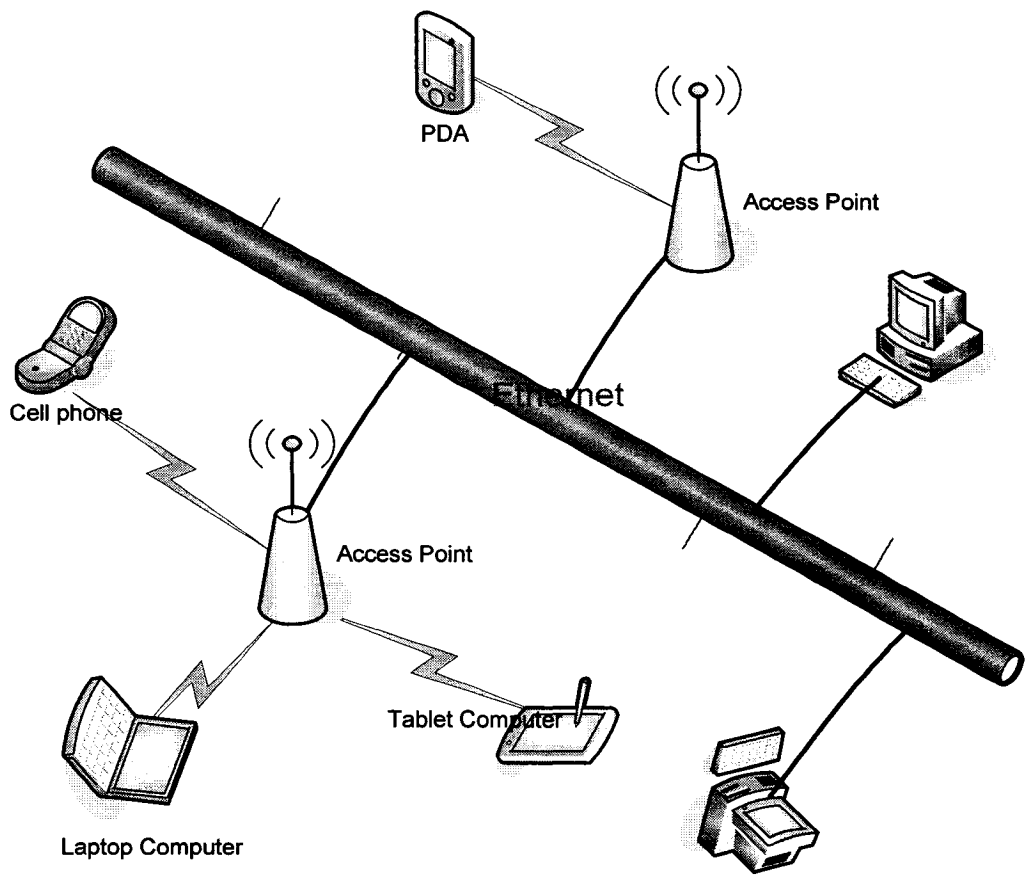

Figure 1.1 Infrastructure-based wireless networks

Ad-hoc based mobile wireless networks (MANETs) are autonomous and self-organizing systems composed of mobile nodes connected by wireless links. Mobile nodes are free to move and organize themselves without the restriction of any fixed infrastructure, such as centralized access points or base stations. Each mobile node in MANETs plays the role of a router and communicates with other mobile nodes. If the source and destination mobile nodes are not within communication range of each other, data packets are forwarded to the destination mobile nodes by relaying transmissions

through other mobile nodes that exist between them. Several nodes may cooperate with each other to carry out a certain task. MANETs are constrained by limited communication bandwidth, frequent network partitioning, and limited battery lifetime [3]. Despite these limitations, MANETs have two attractive advantages. First, MANETs can be deployed quickly because they do not require any fixed infrastructure. Second, MANETs are robust in the face of single point failures since all nodes in the network are equal in status. MANETs are popular in situations such as emergency operations, disaster relief efforts and battlefields because it is impossible or too expensive to deploy infrastructure-based wireless networks in these situations.

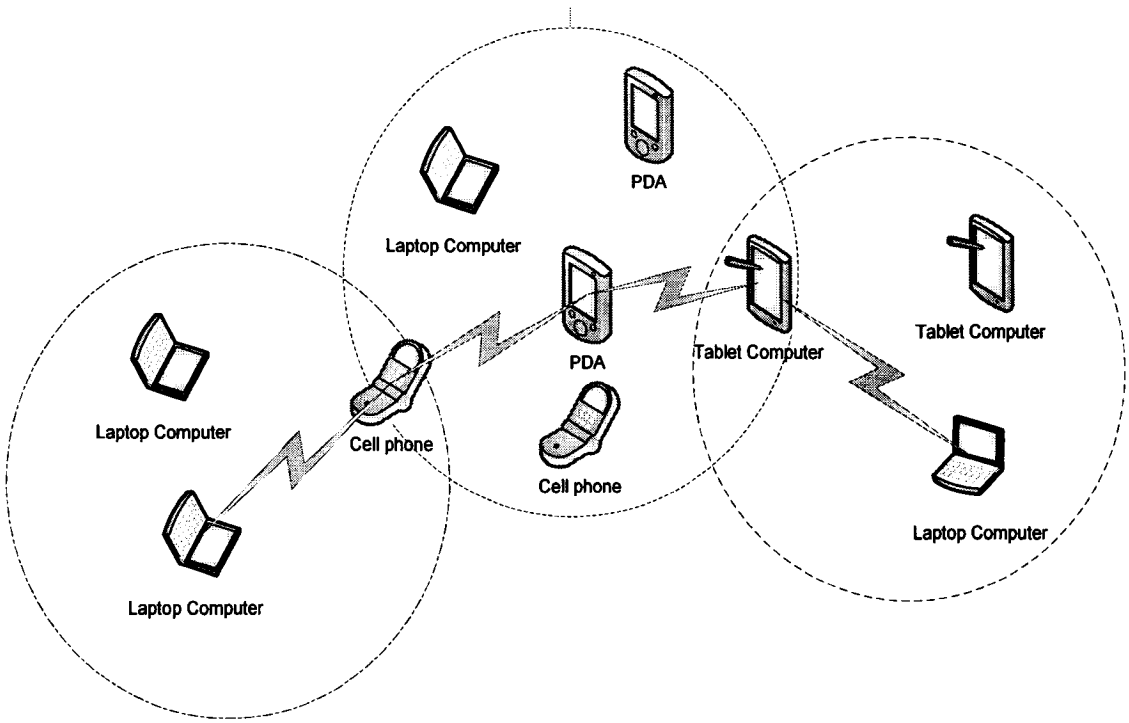

Figure 1.2 Mobile ad-hoc networks

Benefits of data replication include increased data availability, decreased data access time, and balanced workload. Replication, which creates redundant information in a system, also increases system reliability. It allows systems to remain operational in

spite of node and/or link failures. Replication, however, incurs space overhead in nodes and requires complex algorithms to maintain data consistency. Database systems adopt replication techniques to minimize query latency [33, 34]. Peer-to-peer file sharing systems use replication techniques to improve file availability [35].

Trade-offs between consistency, performance and availability in data replication have been well studied [55, 56 and 57]. Replication algorithms can be categorized as static or dynamic. Static replication decides the number of replicas and their locations when data enters the system. This decision does not change during the lifetime of data. Based on data access patterns and system load, dynamic replication changes the number of replicas and their locations from time to time. It is believed that dynamic replication is a feasible solution to alleviate poor network connection and partitioning in mobile environments.

Two important issues in dynamic replication must be addressed. The first issue is replica management, which decides when to create or destroy a replica, and where to place a replica. The second issue is the replica consistency protocol, which decides how to propagate replica updates. Compared with wired networks, data availability in MANETs is low. In MANETs, users move from one place to another place while accessing replicas. Unless replicas move with the user in the form of a cache, they need to be retransmitted back and forth. Retransmission wastes limited communication bandwidth and slows down transaction processing. Moreover, retransmission results in faster consumption of limited battery power. On the other hand, most mobile devices have limited memory and battery so it is impossible to replicate all data on mobile devices. In order to achieve better data access performance and data availability in MANETs, efficient and reliable replication algorithms are important. The performance of

mobile networks is sensitive to the distribution of data among mobile nodes. Reads are usually sent to one node or a set of nodes. It is critical that data is close to nodes that require more reads. On the other hand, writes are usually done on all replicas or a majority of replicas. This requires replicas to be close to each other [58]. In MANETs, replication algorithms need to dynamically adjust to deal with the dynamic network topology. Maintaining replica consistency in MANETs is extremely difficult. The lack of a central control and frequent network partitioning make it difficult to propagate replica updates. It should be noted that creating more replicas improves overall data availability. More replicas, however, make replica consistency more difficult to maintain.

1.2 Objectives

In this thesis, we study dynamic replication algorithms for MANETs. We investigate three questions in dynamic replication: (1) When and where replicas need to be created? (2) When and which replicas need to be destroyed? (3) How to maintain replica consistency? We use simulation to evaluate dynamic replication algorithms, FDRM [7] and ARAM [8], along with our extension of them and a static replication algorithm. FDRM, proposed for wired networks, is a decentralized dynamic replication algorithm with an interesting access frequency detecting mechanism. ARAM is a dynamic replication algorithm in MANETs. In ARAM, mobile nodes dynamically adjust the replica set based on workload and hop distance. MFDRM, our extension of ARAM and FDRM, uses the access frequency detecting mechanism in FDRM and dynamically adjusts the replica set based on workload and hop distance. A major difference between ARAM and MFDRM is that a replica node in ARAM only expands replicas to its neighboring non-replica nodes that read from it, while a replica node in MFDRM expands replicas to non-replica nodes that read from it.

While we acknowledge that most dynamic replication algorithms are not designed for MANETs, studying their behavior in various scenarios yields insights into the design of replication algorithms for MANETs. This thesis makes the following contributions:

- Study and evaluate dynamic replication algorithms in MANETs in terms of availability, consistency, performance and energy consumption.
- Based on FDRM and ARAM, we provide an extension, MFDRM, which uses the access frequency detecting mechanism in FDRM and adopts a mechanism of adjusting replica sets dynamically based on workload and hop distance.
- Provide suggestions and guidelines for the design of replication systems in MANETs.

1.3 Organization

The rest of this thesis is organized as follows. Chapter 2 presents an overview of MANETs and data replication. We also introduce previous work on replication algorithms in MANETs. Chapter 3 presents the details of the dynamic replication algorithms FDRM [7] and ARAM [8]. We also discuss MFDRM, which is our extension of ARAM and FDRM. Chapter 4 presents a comprehensive simulation model. We evaluate and analyze the algorithms discussed in Chapter 3 along with a static replication algorithm under different scenarios. Finally, Chapter 5 presents conclusions and future work.

Chapter 2

Background and Related Work

In this chapter, background information about data replication and MANETs are provided in Section 2.1 and Section 2.2. Related work including previous work of data replication algorithms in wireless networks is discussed in Section 2.3.

2.1 Data Replication

In distributed systems, a data object is often accessed from multiple locations so it is beneficial to replicate data objects through out a network. The benefits of data replication [9] are as follows:

- Increased data availability

  The existence of multiple replicas improves data availability and reliability in the face of network failures.

- Faster query response

  Queries initiated from nodes where replicas are stored can be satisfied directly without incurring network transmission delays from remote nodes.

- Load sharing

  The computational load of responding to queries can be distributed among a number of nodes rather than centralized at a single node.

Achieving these benefits, however, incurs additional cost and complexity in systems. For example, maintaining replica consistency causes severe scalability problems and increases communication costs.

One important aspect in data replication is mutual consistency, that is, all replicas must agree on exactly one current value. When communication fails between nodes containing replicas of the same data, maintaining mutual consistency between replicas becomes extremely complicated. The most disruptive of these communication failures are partition failures, which separate the network into isolated sub-networks called partitions. In general, data replication is an effective way to improve data access performance and data availability where network connectivity is not reliable, and network partitioning may happen at any time. Replication, therefore, provides a way to support data services in MANETs.

2.1.1 Replication Models

A replication model refers to the relationship between replicas. There are three basic replication models [60, 61]: (1) master-slave, (2) client-server and (3) peer-to-peer. The master-slave model (Figure 2.1) assigns one replica as master while other replicas are slaves. The advantage of this model is simplicity. It, however, offers limited functionality, such as slaves are essentially read-only. Replica updates are performed only at the master and slaves synchronize directly with the master.

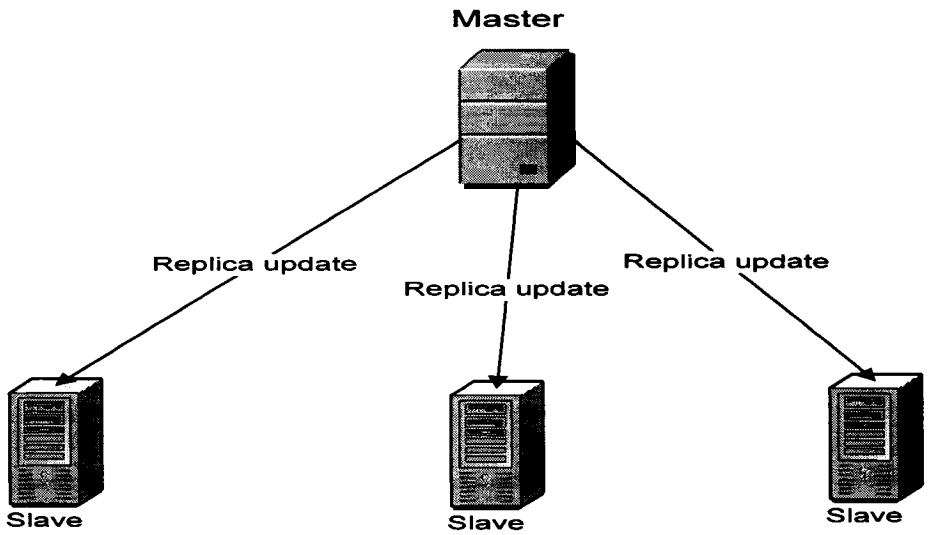

Figure 2.1 Master-slave replication model

The client-server model (Figure 2.2) designates one server to serve multiple clients. The functionality of a client is greatly improved and multiple inter-communicating servers are permitted. Replica updates can be performed at a client. The client-server model has been successfully implemented in replication systems such as Coda [36]. The major drawback of the client-server model is that clients can only communicate with one of the servers. They cannot directly synchronize with each other. This limits their functionality when clients are mobile. For example, two clients in a room cannot exchange updates directly. Both of them must communicate with remote servers. Although multiple servers make client-server models more scalable and robust in the presence of single server failure, the absence of direct communication between clients limits the advantages of data replication.

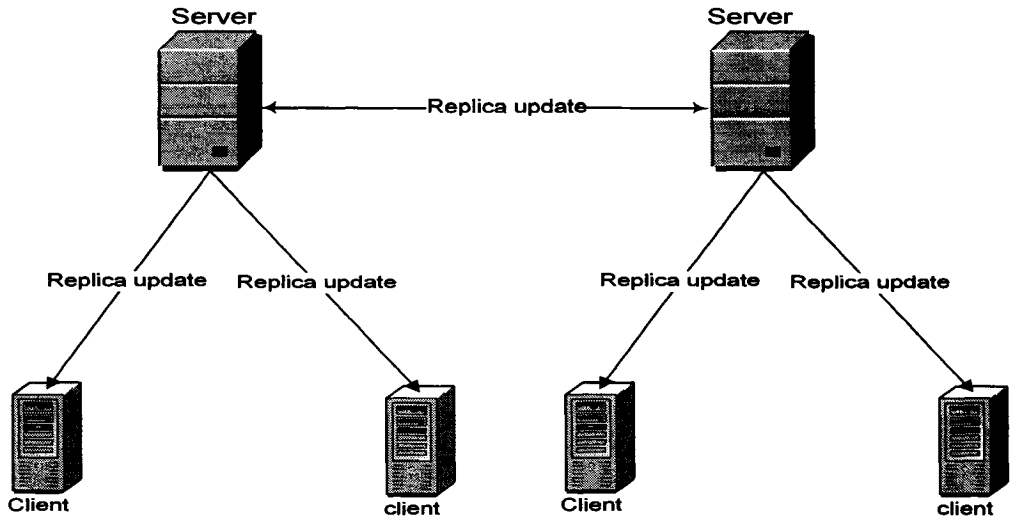

Figure 2.2 Client-server replication model

The peer-to-peer model (Figure 2.3) removes the restrictions in the client-server model. All replicas are equal and replica updates are allowed among all replicas. Replica updates can be propagated to all peers without the requirement that all peers are connected at the same instant. The single point of failure is naturally eliminated. The peer-to-peer model is quite useful in mobile systems that have poor network connectivity. It has been implemented in systems such as Bayou [37], Ficus [38] and Rumor [39]. The peer-to-peer model, however, has poor scalability because each node must store all replica information. Such an approach consumes a large amount of space at each node and additional communication costs are needed to synchronize all replica information.

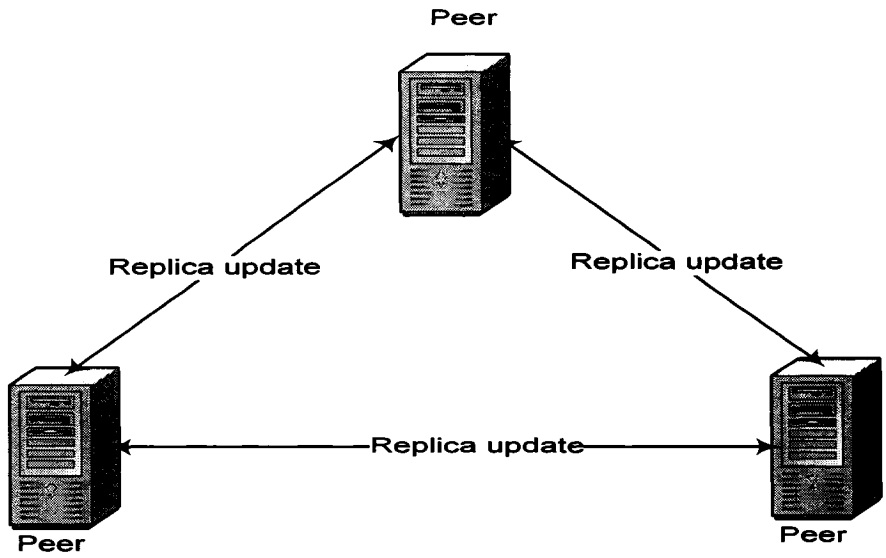

Figure 2.3 Peer-to-peer replication model

2.1.2 Replica Consistency

A major issue in data replication is how to maintain replica consistency among all replicas. From a replica consistency point of view, replication techniques can be classified into two categories: pessimistic replication and optimistic replication [59].

Pessimistic replication keeps replica consistency all the time through various consistency protocols. Pessimistic replication always makes worst case assumptions and prevents update conflicts before they occur. If one replica fails, all replica updates failed. As the number of data replicas increases, the probability of one replica failure increases and the data availability thus decreases. Pessimistic replication is used in banking applications that must avoid giving a wrong answer.

Various pessimistic replica consistency protocols have been proposed in distributed systems [9]. *Read-One-Write-All (ROWA)*, the most popular replica

consistency protocol, states that all copies must be updated and any one copy can be read. The advantage of this method is its simplicity and its ability to process read operations in spite of node or communication link failures as long as at least one node remains up and reachable.

Read-One-Write-All-Available (ROWA - A) [9] is a modified *ROWA* approach. In this method, a transaction is no longer required to ensure updates on all copies of a data item, but only on all available copies. This method avoids the delay incurred by an update transaction when some nodes fail. For this protocol to work properly, failed nodes are not allowed to become available again until they recover by copying the current value of a data item from an available copy. *ROWA - A* can tolerate $n-1$ node failures, but it does not tolerate network partitioning.

Another interesting variant of *ROWA* is called *Primary Copy ROWA* [9]. In this method, a specific copy of a data item is designated as the primary copy and the remaining copies are backup copies. A replica update is carried out at the primary copy which forwards the update to all operational backups while reads are satisfied from either primary copy or backup copies. A transaction that updates a replica is allowed to commit only after the primary copy and all operational backup copies have successfully recorded the update operation. When the primary copy fails, a backup copy is chosen through an election protocol to take its responsibility as the new primary. This requires that failure of the primary site is detectable and distinguishable from failure to communicate with it. However, such a distinction is theoretically impossible [40, 41].

The *ROWA* family of protocols implicitly favors read operations by allowing a read to proceed with only one copy, while requiring write operations to be carried out at all up

nodes. This latter condition also means that *ROWA* algorithms can not permit write operations to succeed when it is not possible to communicate with an up node because of network failures. These drawbacks gave rise to the quorum consensus *(QC)* approach [9]. *QC* methods, often termed *voting* methods, allow writes to be recorded at only a subset (a write quorum) of the up nodes, so long as reads are made to a subset (a read quorum) that is guaranteed to overlap the write quorum. This quorum intersection requirement ensures that every read operation returns the most recently written value. Different *QC* methods adopt different styles for specifying quorum membership. Quorums can be *static* as they are specified by votes that are assigned once and for all at system startup time. Quorums also can be *dynamic* when the nodes are capable of reconfiguring the quorum specification.

In the uniform majority QC method [9], the distributed system is modeled as a group of sites that vote on the acceptability of read or write requests. A read operation or a write operation succeeds if and only if a majority of the sites approve its execution. Not all sites that vote for an operation need to carry it out on their local copies. In particular, a read operation needs to be executed at only one current copy while a write operation must be executed at a majority of the copies. The majority requirement ensures that there is an intersection between the read and write operations. This method is resilient to both site and network failures, but it has high read and write costs. At least half of n sites must participate, via voting, in every data access. This method works on the presumption that a network failure partitions the sites into two groups - a majority partition and non-majority partition. Repeated failures may, however, split the system into many groups of sites, with none of the groups forming a majority.

The weighted majority *QC* method [10] generalizes the notion of uniform voting. Instead of assigning a single vote per site, each copy of a data item *d* is assigned a non-negative weight (a certain number of votes) whose sum over all copies is *u*. The data item *d* itself is assigned a read threshold, denoted by *r*, and a write threshold, denoted by *w*, such that: r + w > u and w > u / 2. A read (or write) quorum of *d* is any set of copies with a weight equal to at least *r* (or *w*). This constraint of read and write thresholds provides greater flexibility in vote assignment while ensuring mutual consistency of the copies just as in majority consensus. A versioning mechanism determines the currency of the copies. Because of the need to poll multiple sites before every read operation, workloads with a high proportion of reads do not perform well under this method. The quorum consensus algorithm does not require a complicated recovery protocol. A copy of the data item *d* that is down and therefore misses some writes does not have the largest version number in any read or write quorum of which it is a member. Upon recovery, it cannot be read until it has been updated at least once. Transactions continue to ignore stale copies until they are brought up-to-date. The algorithm is flexible. By altering *r*, *w*, and the weight assignment to the copies of each data item in replication systems, the performance characteristics and the availability of the protocol can be altered.

Optimistic replication allows inconsistency among replicas in a controlled way [59]. They propagate updates in the background and allow any read or write operation to be performed on a replica as long as any one of the replicas is accessible. In general, optimistic replication assumes that concurrent updates and conflicts are rare and most conflicts can be resolved transparently without user involvement. Optimistic replication greatly improves data availability at the expense of increasing the risk of serving stale data.

2.1.3 Replica Management

Replica management algorithms, which decide the number of replicas and their locations, affect the performance of distributed systems [64]. For example, reading from a local cache is faster than reading from a remote node. In general, for read-intensive data a widely distributed replica management algorithm is preferred since it increases the number of local reads and balances workload among replica nodes. On the other hand, a replica update is usually written to all replicas or a majority of all replicas. For write-intensive data, a narrowly distributed replication algorithm is therefore preferred.

There are two kinds of replica management algorithms: static replica management algorithms and dynamic replica management algorithms. A static replica management algorithm decides the number of replicas and their locations when a data object enters the system and this decision does not change during the lifetime of the data object. The decision is usually based on access patterns. The static approach performs well if access patterns are known and fixed, but this is not always possible especially in a large scale distributed system environment.

A dynamic replica management algorithm makes its decision when a data object is created and changes the number of replicas and/or their locations during the lifetime of a data object. Dynamic replica management algorithms can be classified into two types: centralized algorithms and distributed algorithms. A centralized algorithm has a global view of the access patterns of all replicas and makes its decision based on the information that is gathered through a central point. The benefit of a centralized algorithm is its simplicity. A centralized algorithm, however, is likely to become a bottleneck for a

large distributed system. In a distributed algorithm, a node makes its decision to add or delete replicas based on its local information and need not know about replicas on other nodes.

Distributed algorithms have two advantages over centralized ones. First, they respond to changes in the read-write pattern in a timelier manner because they avoid the delay involved in the collection of statistics and computation. Second, their overhead is lower because they eliminate extra message exchanges required in centralized algorithms. A distributed algorithm, therefore, is more likely to be implemented in large systems since it does not require any centralized point. However, coordination is required among different decision-making nodes in order to agree on a replication decision. In general, a distributed solution that requires limited coordination among all decision-making nodes is preferred since it produces less overhead on the system.

Acharya and Zdonik [13] present an efficient dynamic replication system. The aim of their scheme is to optimize message traffic by intelligent data placement. Each node maintains a *deterministic finite state automaton* for every neighbor node to record each neighbor's access pattern for each file. It uses a finite automaton-based technique to learn access patterns. This acquired information is then used to predict future access sequences and dynamically reorder replicas or delete replicas. The algorithm is adaptive and distributed in nature. It continuously adjusts the replication scheme and adapts to changes in access patterns. Each node decides to add or delete a replica based on its local information such as statistics about reads and writes. This algorithm adapts well to the system and has good performance, but it is complicated to construct the deterministic finite state automaton and it needs to maintain a large amount of

information for both replicas and neighbors. The computational complexity increases dramatically when the system scale increases.

Giacomo [12] has another interesting solution for dynamic replication, in which a node scans its replicas periodically. When it finds the read-write ratio of a replica reaches a certain level, it invokes the dynamic replication algorithm to add or delete replicas. The solution only involves replica nodes and is easy to implement. Simulation results, however, demonstrate that the system performance is sensitive to the scanning periodicity. If the file is not accessed frequently, keeping a small scanning period is wasteful and leads to an unnecessary load on the node. If the file is intensively accessed, a large scanning period may not respond to change in a timely manner.

Bartal [11] presents another algorithm for dynamic replication. Each node maintains a counter C for each file it accesses. The value of C goes up with a read of the file. When C reaches a certain value D, the node adds a new replica to the scheme. The counter C also goes down with a write of the file. When the counter drops to zero, the node deletes its replica. Bartal's algorithm is simple to realize, but it is difficult to determine a proper value for D, and all nodes accessing a file have to maintain a counter of the file whether the node has the replica or not. This obviously enlarges the range of managed nodes.

2.2 Mobile ad-hoc networks

The popularity of mobile devices along with the presence of ad-hoc networks has contributed to recent advances in the field of mobile computing in ad-hoc networks [63]. MANETs consist of mobile nodes, which form a temporary network without the aid of an established infrastructure or centralized administration. MANETs can be employed in

areas such as emergency rescue sites, combat fields, etc. Each mobile node in a MANET acts as a node and as a router. Mobile nodes can forward packets on behalf of each other. If a source node cannot send a packet directly to a destination node due to limited transmission range, the source node sends packets to intermediate nodes, which forward packets toward the destination node. According to Johnson [44], a MANET is defined as follows:

"An ad-hoc network is a collection of wireless mobile hosts forming a temporary network without the aid of any centralized administration or standard support service regularly available on the wide-area network to which the hosts may normally be connected."

2.2.1 Characteristics of MANETs

The characteristics of MANETs are summarized as follows:

1. **Dynamic Network Topology**

   Most nodes in an ad-hoc network are wireless mobile devices. They move, join and leave the network freely. Ad-hoc networks can be formed wherever mobile nodes are located since they do not require a physical infrastructure. They can also assemble and disassemble on demand. This feature makes the network topology change unpredictably. There are many reasons that cause the network topology to change. For instance, each node in an ad-hoc network can arbitrarily decide to join or leave the network. A node may also be disconnected from the network when it accidentally moves out of wireless transmission range, runs out of power, or gets unexpected transmission inference. Even though the network topology varies, connectivity in the network should be maintained to allow applications and services to operate without disruption. This characteristic affects the design of routing protocols.

2. **Bandwidth and energy constraints**

Wireless links have limited bandwidth. Moreover, since wireless links have lower capacity than wired links, traffic congestion is typical rather than exceptional. Besides, the throughput of wireless communications in real environments is often much less than a radio's maximum transmission rate because of the effects of multiple access, fading, noise, and interference conditions [45]. Ad-hoc network applications must minimize various network overheads. Most nodes in ad-hoc networks are battery powered [22]. Battery life is limited; therefore protocols and applications need to be energy efficient. Energy conservation is considered a key system design criterion.

3. **Constrained Network Security**

MANETs are vulnerable to attack and there is an increased possibility of spoofing and denial-of-service attacks.

4. **No fixed network infrastructure**

MANETs were initiated as a means of communicating in the battlefield where network infrastructure is not always available or an infrastructure can be easily interfered with by an enemy. One important goal of ad-hoc networks is not to rely on any form of infrastructure for support in routing, network management and data transmission. All nodes are equipped with packet forwarding capabilities. Each node in ad-hoc networks can communicate with its peer directly in a single hop, or in multiple hops, without using any existing infrastructure. These features significantly reduce the cost of deploying, maintaining and administering the network.

5. **Decentralized Network Control**

MANETs are very robust in the face of single-point failures since all nodes in the network are equal in status.

2.2.2 Routing Protocols in MANETs

In MANETs, two nodes may not be able to communicate directly with each other because they are out of radio transmission range. In Figure 2.4, node *C* is not within the transmission range of node *A* and node *A* is not within the transmission range of node *C*. If node *A* and *C* want to communicate with each other, they must ask node *B* to forward packets for them because node *B* is within the transmission ranges of both nodes *A* and *C*. In a real ad-hoc network, routing problems are more complex than this example, because any or all nodes associated with the network may move at any moment.

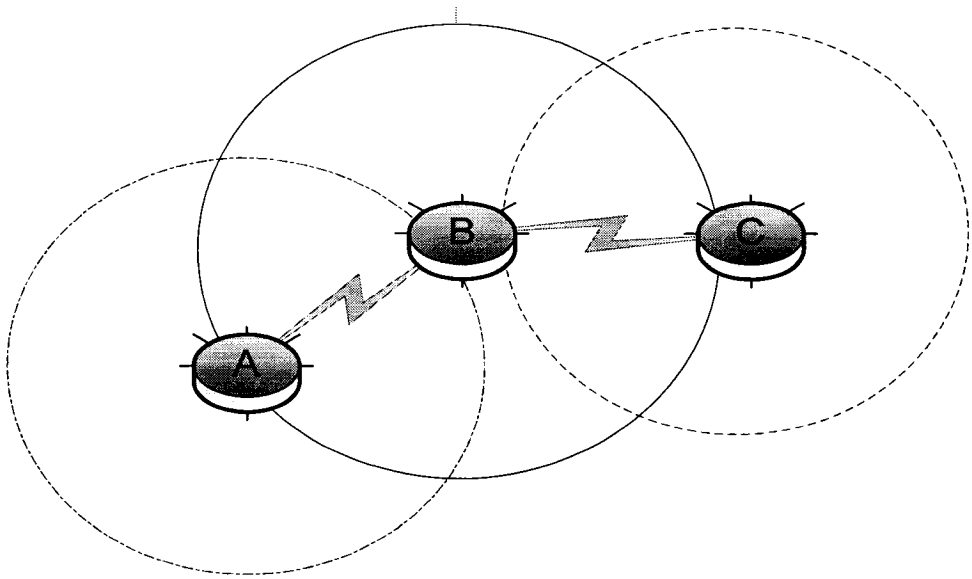

Figure 2.4 A MANET with three mobile nodes

Many routing protocols in MANETs derive from the distance vector or link state algorithms [21]. In distance vector routing, each router maintains a table containing the distance from itself to possible destinations. Each router periodically transmits this table information to its neighbor routers, and updates its own table by using the values received from its neighbors. Based on the comparison of the distances obtained from its

neighbors for each destination, a router can decide the next hop which gives the shortest path from itself to the destination. When the routing table is frequently updated, the algorithm speeds up the convergence to the correct path. However, the overhead in CPU time and network bandwidth for flooding routing updates also increases.

In link state routing, each router has a complete view of the whole network topology. Each router checks the cost of the link to each of its neighbor routers and periodically floods updated information to other routers in the network. After each router receives this update of the cost of each link in the network, it calculates the shortest path to each possible destination. When a router needs to forward a packet to a destination, it transmits the packet to the next hop router based on the best path to the destination acquired from the updated information. Link state routing protocols show faster convergence to the correct network view when the network is dynamic. However, compared to the distance vector routing protocol, a link state routing protocol requires more CPU time for computing the complete shortest route to each possible destination, and more network bandwidth for broadcasting the routing updates from each router to all other routers in the whole network.

Routing protocols in MANETs can be also classified into reactive routing protocols and proactive routing protocols [49]. Reactive routing protocols do not initiate route discovery until it is needed. In other words, route discovery is totally on-demand. Examples of reactive routing protocols for ad-hoc networks include Ad-hoc On- Demand Distance Vector (AODV) [48], Dynamic Source Routing (DSR) [50], On-Demand Multicast Routing Protocol (ODMRP) [51] and Temporally-Ordered Routing Algorithm (TORA) [52]. On the other hand, proactive routing protocols are based on the periodic exchange of network topology knowledge. The proactive protocols provide a needed

route instantly at the expense of bandwidth because of transmitting periodic updates of topology. Examples of proactive routing protocols include Destination-Sequenced Distance Vector (DSDV) [46], Source-Tree Routing (STAR) [53] and Topology Dissemination Based on Reverse-Path Forwarding (TBRPF) [54].

Perkins and Bhagwat [46] devised a Destination-Sequenced Distance Vector (DSDV) protocol in MANETs. It is based on the famous Bellman-Ford routing algorithm. DSDV also has the features of a distance-vector protocol since each node holds a routing table including the next hop information for each possible destination. A performance study on DSDV [48] shows that it is able to deliver virtually all data packets when each node moves with relatively low speed. However, as the mobility of each node increases, the speed at which the system converges to the correct path decreases.

The Dynamic Source Routing (DSR) [50] protocol is based on source routing, which means the source of a packet determines the complete route from itself to the destination. All intermediate nodes along the route simply forward the packet to the next hop indicated in this predetermined route. No routing decision is made at intermediate nodes. The advantage of source routing is that intermediate nodes do not need to maintain up-to-date routing information in order to route packets because the packets already contain all routing decisions. The obvious disadvantage is that data packets must carry predetermined routes.

DSR consists of two major operations: route discovery and route maintenance. Each node maintains a cache of routes it has learned so far, called the route cache. When a node attempts to send a data packet to a destination, it first checks its route cache to determine whether it already has a route to the destination. If an unexpired

route to the destination is found, the node uses this route to send the packet. Otherwise, the node initiates a route discovery operation to discover a route. Route discovery works by broadcasting *Route_Request* packets. A *Route_Request* contains the address of the destination as well as a *Route_Record* that records the nodes that the request has passed by. Each node receiving a *Route_Request* checks whether it knows a route to the destination, i.e. the desired route is contained in its route cache, or it is itself the destination. In both cases, the complete route from the initiator to the destination is found. A *Route_Reply* packet, which includes the route, is forwarded to the initiator. Otherwise, the node appends its own address to the *Route_Record* of the route request and re-broadcasts the route request to its neighbors.

Route maintenance in DSR is invoked when a route is broken. Routes may become invalid due to node movement. To quickly adapt to this change, each node constantly monitors its links. If a node in a route finds out that it cannot forward packets to the next node in the route, it immediately sends a Route_ Error packet to the source of the route. Therefore, the source is able to quickly detect an invalid route and stop using it. The source removes any route using this link from its cache. A new route discovery process must be initiated by the source if this route is still needed.

Ad-hoc On-demand Distance Vector (AODV) [47] is a reactive protocol which combines both DSR and DSDV characteristics. It adopts the route discovery and route maintenance mechanisms in DSR. It also uses mechanisms in DSDV, such as hop-by-hop routing, sequence numbers and beacons. AODV does not maintain any routing information nor transmit any periodic advertisement packets for exchanging routing tables. Route Discovery is performed only when a new route is desired. In other words, only when two nodes need to communicate with each other will they exchange routing

packets to maintain connectivity between them. AODV also depends upon dynamically establishing route table entries at intermediate nodes.

AODV also consists of two procedures: route discovery and route maintenance. The route discovery process occurs when a source node wishes to transmit traffic to a destination node to which it has no route. The source node generates a route request (RREQ) message that is flooded in a limited way to other nodes. A route is considered found when the RREQ message reaches either the destination itself, or an intermediate node with a valid route entry for the destination. A route reply (RREP) message is unicasted back to the originator of a RREQ. The reason one can unicast the message back, is that every route forwarding a RREQ caches a route back to the originator. As long as a route exists between two endpoints, AODV remains passive. When the route becomes invalid or lost, AODV will again issue a route request.

In AODV, route maintenance is designed to deal with the problem of frequent broken links and route failures. In order to monitor the link status of next hops in active routes, a node periodically broadcasts HELLO messages to its direct neighbors to determine whether the neighbors are alive. When a link breakage in an active route is detected, a route error (RERR) message is used to notify other nodes of the loss of the link. In order to enable this reporting mechanism, each node keeps a list, containing the IP address for each of its neighbors that are likely to use it as a next hop towards each destination.

Figure 2.5 illustrates an AODV route lookup session. Node A wishes to initiate traffic to node H, to which A has no route. Node A broadcasts a RREQ, which is flooded to all nodes in the network. When this request is forwarded from D from H, H generates a

RREP. This RREP is then unicasted back to A using the cached entries in node D.

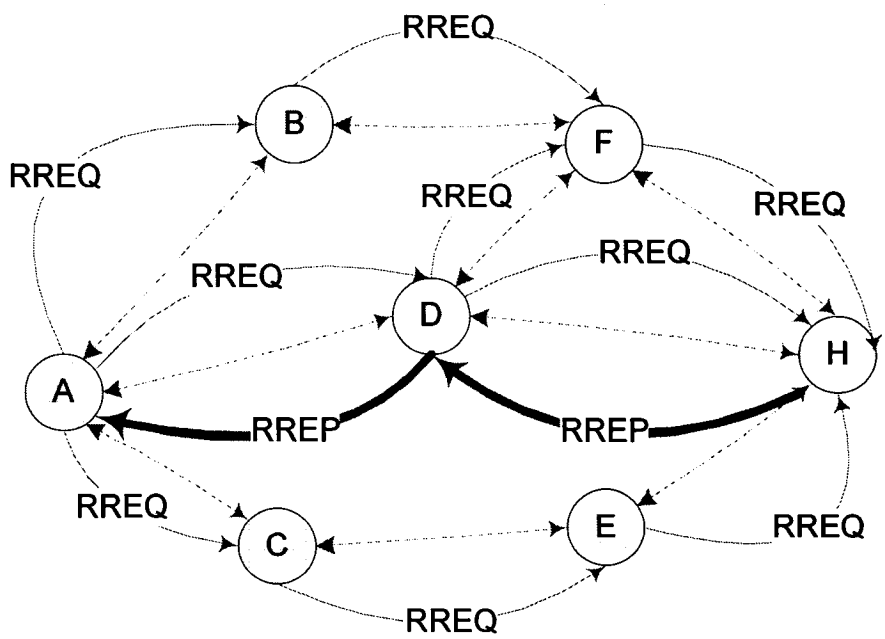

Figure 2.5 An AODV route discovery session

2.3 Data replication in wireless mobile networks

2.3.1 Data replication on infrastructure-based wireless mobile networks

Recent advances in hardware technologies such as portable computers and wireless communication networks have led to the emergence of mobile computing systems. Data replication is an important technique that reduces contention on the narrow bandwidth wireless channel. Replication in mobile wireless networks, however, is severely affected by disconnection and nodal mobility. Barbara [14] points out that even though the number of replicas in mobile networks can be large, the key issue is how to manage replicas that can be updated. Several strategies [16, 17, 18 and 19] have been proposed for replication or caching in traditional infrastructure-based wireless mobile networks. Most of these strategies assume that mobile nodes access data at a base

station and replicate or cache data on mobile nodes. These strategies focus on reducing the one hop wireless communication cost induced by keeping data consistency between base station and mobile nodes.

It is important that mobile nodes access data in a way that minimizes communication because of limited wireless bandwidth. This can be achieved in a one-copy or two-copy replica allocation scheme [62]. In a one-copy scheme only the base station stores data whereas in a two-copy scheme both the base station and the mobile nodes store data. For example, if reads from a mobile node x to a data item n, located at the base station, are frequent while updates from the base station are infrequent, then it is beneficial to allocate a replica of n to x. This way reads from x access its local replica, and do not require additional communication. The infrequent updates are transmitted from the base station to x. In contrast, if reads from x are infrequent compared to writes from the base station, then n should not be allocated to x and reads should be sent to the base station.

Sistla, Wolfson and Huang [20] propose a family of dynamic data replication methods in infrastructure-based mobile networks. Based on the read-write ratio, these methods select different allocation schemes. If reads are more frequent, the two-copy scheme is selected. Otherwise, the one-copy scheme is adopted. Based on a sliding window of k requests, their methods allocate or de-allocate data items to mobile nodes. For the latest k requests, if the number of reads is higher than the number of writes and the mobile computer does not have a copy, then a copy is allocated to the mobile computer. If the number of writes is higher than the number of reads and the mobile computer does have a copy, then the copy is de-allocated. Thus, the allocation scheme

is dynamically adjusted according to the relative frequencies of reads and writes. Algorithms in this family differ on the size of the window.

2.3.2 Data replication in MANETs

Little research work has been done to provide data replication in MANETs. Existing solutions for data replication in traditional one-hop wireless networks is not applicable to MANETs. These solutions are not designed for dynamic networks. The high possibility of network partitioning in MANETs means that some nodes may not be able to communicate updates to other nodes or may not be able to retrieve the latest information on queries.

Hara [4] proposes three replication methods that take into account data access frequencies and network topology. The goal of these algorithms is to improve data accessibility in MANETs. The author assumes replicas are relocated in a specific period, called the relocation period. Replica allocation is determined based on the access frequency from each mobile node to each data item and the network topology. The author makes two major assumptions. First, data items cannot be updated. Second, the access frequencies to data items from each mobile host are known and do not change. The algorithm is periodically executed to cope with the dynamically changing network topology. The three methods proposed in [4] are described as follows:

- **Static Access Frequency Method (SAF)**

  Only the access frequency to each data item is taken into account. Each mobile node allocates replicas in descending order of the access frequency.

- **Dynamic Access Frequency and Neighborhood Method (DAFN)**

Both the access frequency to each data item and the neighborhood among mobile hosts are taken into account. Replicas are initially allocated based on the SAF method, and then replica duplication is eliminated among neighboring mobile nodes

- **Dynamic Connectivity Based Grouping (DCG) method**

  The access frequency to each data item and the whole network topology are taken into account. An algorithm is executed to find bi-connected components in the network. Bi-connected components form a stable group. In each component, the cache spaces of all nodes are treated as a big cache. Replicas are allocated into the big cache according to the access frequency.

Simulation results show that in most cases, the *DCG* method gives the highest accessibility and the *SAF* method generates the lowest traffic.

Hara [5] extends the three methods to *E-SAF, E-DAFN* and *E-DCG* by allowing periodic updates to data items. Although reducing the degradation of data accessibility to some extent, both scheme *DCG* and scheme *E-DCG* have a major drawback in that they require all data nodes to broadcast their information to other nodes which causes a significant amount of network traffic. This situation is more severe in MANETs because of limited network bandwidth.

Huang and Chen [23] explore data replication in MANETs with group mobility. In reality, the moving behavior of mobile nodes is usually regular and follows some mobility patterns [24, 25]. Group mobility usually occurs in collaborative works. For example, a group of visitors visiting an art gallery with the same guide usually have similar movement behavior. The underlying group mobility model is assumed to be the

Reference Point Group Mobility model (RPGM) [26]. To avoid blind flooding [27], their scheme takes a bottom-up approach without requiring global network connectivity. Each mobile node first exchanges its motion behavior with its neighbors. The coverage of the information exchange is limited by a predetermined parameter. A decentralized clustering algorithm is proposed to cluster mobile nodes with similar motion behavior into mobility groups. Finally, data items are replicated according to the resulting allocation units. Moreover, the scheme maintains the mobility groups in an adaptive manner, which minimizes the number of information broadcasts.

Karumanchi [28] proposes a variation of the quorum-based approach to support data replication in MANETs, with the assumption that there are many designated servers throughout the network. These predefined servers are divided into a number of quorums. Each quorum has at least one server in common. Updates are sent from a node to a quorum of servers and a mobile node queries a data item from a quorum of servers to get the most up-to-date data. The reaching ability of these predefined servers changes when the network topology changes. Therefore some queries may fail. Their work developed heuristics for clients to select servers with the highest likelihood of being accessible in order to maximize the chance of successful replica query.

Wang and Li [6] propose an algorithm to predict network partitioning and allocate replicas before the network partitioning happens to ensure replica availability. Their approach ensures replica availability to clients by dynamically creating and placing servers based on the changing network topology. The solution utilizes observed node mobility patterns to predict the occurrence of partitioning, and takes necessary actions in advance to efficiently provide continuous service availability when the network partitioning happens. On the servers, the authors also present a simple sequential

clustering algorithm that can identify correlated mobility patterns, which are used to predict the time and location of network partitioning. On the clients, a fully distributed grouping algorithm is proposed. The grouping algorithm discovers mobility group membership based on the stability with respect to distances of neighboring nodes.

Ratner [42] proposes an optimistic replication system called ROAM. Mobile nodes in ROAM close to each other are grouped into domains. One of the mobile nodes inside the domain is chosen as a master node. The master node is in charge of communication with other domains. All master nodes form an adaptive ring and update operations are propagated epidemically along this ring to each domain. Yu, Martin and Hassanein [43] propose a novel optimistic replication scheme, called Distributed Hash Table Replication (DHTR). DHTR organizes all mobile nodes into non-overlapping clusters and builds a two-level distributed replica information directory on cluster heads to facilitate the propagation of query and update messages. DHTR also employs distributed hash table techniques to speed up the directory lookup process. Although both seek to improve the performance of replica updates, their studies do not address availability and the effects of changing the number of replicas. In addition, construction and maintaining an adaptive ring or a distributed hash table which has a tree structure in MANET impose a lot of load on the network.

2.4 Summary

This chapter provides background information about data replication and the basic characteristics of MANETs. Previous research on data replication in wireless networks is also addressed. Section 2.1 gives an overview of data replication in distributed systems. Replication models, such as the master-slave replication model, client-server replication model and peer-to-peer replication model, are discussed.

Replication consistency protocols, such as Read-One-Write-All and Quorum Consensus protocols are introduced in detail. Previous research on replica management algorithms are addressed as well. Section 2.2 gives a brief overview of MANETs. The characteristics of MANETs are illustrated and different routing protocols, such as DSDV, AODV and DSR are introduced. Section 2.3 presents previous research work in the field of data replication on infrastructure-based wireless networks and MANETs.

Only a few replication algorithms have been proposed for MANETs, to the best of our knowledge and no work has been done to define the guidelines of replication in MANETs. Most existing replication algorithms in MANETs emphasize data availability issues while neglecting performance issues. Moreover, many of these algorithms assume that data are read-only. This obviously limits the functionality of replication systems. In the next chapter, we discuss dynamic replication algorithms, namely FDRM and ARAM, and MFDRM, which is our extension of FDRM and ARAM. The aim of these algorithms is to improve data access performance and to enhance data availability and consistency.

Chapter 3

Dynamic Replication Algorithms

In this chapter, we discuss dynamic replication algorithms, namely FDRM [7] (dynamic distributed replica management mechanism based on accessing frequency detecting), ARAM [8] (dynamic adaptive replica allocation algorithm in MANETs) and MFDRM which is our extension of FDRM and ARAM. FDRM is a dynamic replication algorithm with an access frequency detection feature. Each replica node in FDRM scans its local replica to detect its access frequency and make a decision to add another replica and/or delete the current replica. In addition, the scanning interval of a replica is variable according to the access frequency. ARAM is another dynamic replication algorithm proposed for MANETs. Each replica node in ARAM collects access requests from other nodes and makes decisions on whether to add replicas to neighboring nodes and/or delete its local replicas. MFDRM is our extension of FDRM and ARAM. MFDRM uses the access frequency detecting technique in FDRM. Based on the workload and the hop distance of each data access, each replica node makes the decision locally to add, and/or delete replicas. Section 3.1 outlines our assumptions and definitions for the three algorithms. Section 3.2 and section 3.3 discusses the details of FDRM and ARAM. Section 3.4 discusses the details of our extension, MFDRM. Section 3.5 presents our summary of dynamic replication algorithms on MANETs.

3.1 Assumptions and Definitions

FDRM, ARAM and MFDRM use the ROWA (Read-One-Write-All) protocol to maintain replica consistency. Reads are satisfied from the local cache or from the closest

replica node, while writes are propagated to all replica nodes. The read-write pattern during a time period is predictable based on the read-write pattern in the immediately preceding time period. The shortest hop distance between any two mobile nodes is available. Replica accesses are independent and the cost of multiple accesses is the sum of each individual accesses. Each mobile node has enough power to send and receive packets.

All algorithms work in two phases, namely a data access phase and a replica allocation phase. In the data access phase, each node performs reads and writes based on its local view of replicas. In the replica allocation phase, a node adds and/or deletes replicas.

3.2 The FDRM Algorithm

FDRM is a decentralized, dynamic replication algorithm for wired networks with the goals of reducing network traffic and improving system performance. Each replica node scans the local replica access information and makes decisions independently to add, and/or delete a replica in a periodic manner. The scan interval of each replica varies according to the access frequency of that replica. Table 3-1 lists the definitions of variables that a replica node k maintains.

Notation	Definition
R_{in}	the number of reads from a replica node *k* itself during a time interval T_n
$R_{out,j}$	the number of reads from a non-replica node *j to node k* ($j \neq k$) during a time interval T_n
W_{out}	the number of updates from other nodes during a time interval T_n for a replica node *k*
W_{in}	the number of writes from replica node *k* itself during a time interval T_n
α	system cost of reading a replica
β	system cost of updating a replica
F	the set of replica nodes in the system
V	the set of all nodes in the system
T_n	the time duration of the nth replica allocation interval
A_n	the number of data accesses during the time interval T_n for a replica node *k*

Table 3-1 Notation in FDRM

3.2.1 Replica allocation in FDRM

When a replica is added to a non-replica node *j* ($j \in V$, $j \notin F$), the reads from node *j* can be satisfied from its local cache therefore the overall read cost decreases. On the other hand, writes must update the replica at node *j* therefore the overall write cost increases. The criterion for a replica node *k* to add a new replica to another non-replica node *j* is that the overall read cost decrement is greater than the overall write cost increment. The condition of replica addition in a replica node *k* ($k \in F$) is expressed as in (3.1). FDRM assumes $\alpha \approx \beta$ in a real system and the local read and write costs are zero.

$$R_{out,j}\,\alpha > (W_{out} + W_{in})\,\beta \Rightarrow R_{out,j} > (W_{out} + W_{in}) \qquad (3.1)$$

If a replica node k deletes a replica from itself, the reads from node k must be satisfied from another replica node m, therefore the overall read cost increases. On the other hand, writes do not need update the replica at node k therefore the overall write cost decreases. The criterion for replica node k to delete its replica is that the overall read cost increment is less than the overall write cost decrement. The condition of replica deletion in a replica node k ($k \in F$) is expressed in (3.2)

$$R_{in}\,\alpha < W_{out}\,\beta \Rightarrow R_{in} < W_{out} \qquad (3.2)$$

3.2.2 Interval of replica allocation

A replica node decides whether to add and/or delete a replica after a period of time T. The length of T is important to the system's performance. *FDRM* uses a varied interval, which is sensitive to the frequency of replica access. The interval becomes shorter when the access frequency increases and longer when the access frequency decreases. Different nodes in the system may thus have different replica allocation frequencies. We assume that there is a sequence of intervals T_1, ..., T_n for a replica. The unit of T_i is seconds. A_i denotes the number of accesses during the time period T_i. If the current time interval is T_n, the next time interval T_{n+1} can be calculated as in (3.3)

$$T_{n+1} = T_n \bullet \frac{A_{n-1}/T_{n-1}}{A_n/T_n} \qquad (3.3)$$

3.2.3 Details of FDRM

Appendix A shows the details and pseudo code of FDRM. The current number of replicas in the system, denoted |F|, must be within a certain range. If it is too small or shrinks to zero, the availability of data cannot be guaranteed. If it is too large, then too

many system resources are consumed. FDRM, therefore, sets two thresholds, *MIN* and *MAX*, where $MIN \leq |F| \leq MAX$.

Each replica node k maintains three counters R_{in}, W_{in}, W_{out} and a list *outRead*. Each element in *outRead* has two fields: (1) node identifier j $(j \neq k)$ and (2) $R_{out,j}$ which denotes the number of reads that replica node k received from non-replica node j. The elements of *outRead* are in decreasing time order, which means the first element in the list has the information about the node that most recently accessed the replica. It is believed that the most recently accessed node is the most probable node to have further read access.

When a read request from non-replica node j arrives at replica node k, replica node k first finds whether node j has accessed the replica node k before by searching the *outRead* list. If node j has not accessed replica node k, replica node k inserts a new element at the beginning of the list and sets $r_{out,j}$ to 1 and node id to j. If node j has accessed replica node k before, replica node k increases $r_{out,j}$ by 1 and moves the element to the head of the list.

Replica node k records the current time interval length T_n and the number of data accesses A_n that replica node k received in T_n. Replica node k also records previous interval length T_{n-1} and the number of data accesses A_{n-1} that replica node k received in T_{n-1}. The next interval T_{n+1}, therefore, can be calculated in accordance with equation (3.3).

FDRM also considers some special cases for an allocation interval. If there is no data access during T_n, then no matter how many data accesses occurred during T_{n-1}, T_{n+1} equals $2T_n$. If there is no access during T_{n-1}, then no matter how many accesses occurred in T_n, T_{n+1} equals to $1/2\ T_n$. To avoid the interval being too short or too long, FDRM set

two thresholds *MIN_PERIOD* and *MAX_PERIOD*, where $MIN\_PERIOD \leq T_n \leq MAX\_PERIOD$.

A replica node *k* has six states during the replica allocation phase, namely *Start, Adding, Added, Deleting, Deleted and Switch*. Figure 3.1 illustrates these states. First, a replica node k tries to add a replica to the current replication scheme. It searches *outRead* to find a node *P* which satisfies condition (3.1). If the search for node *P* succeeds, replica node *k* changes its state from *Start* to *Adding*. Next, replica node *k* sends coordination messages *RPCA_ADDING* to other replica nodes to request permission to add a replica. Replica nodes send permission messages back to node *k* only when its state is *Adding*. We denote the number of permission messages that replica node *k* receives in the *Adding* state as *ARP* and the current number of replicas as *CRP*. If $CRP + ARP < MAX$, which means replica node *k* has enough permission to add a new replica, replica node *k* changes its state from *Adding* to *Added*. Otherwise, it changes its state from *Adding* to *Start*. When replica node *k* is in the *Added* state, it sends data message *RPCA_ADD* to node *P*. When node *P* receives the message *RPCA_ADD*, it adds a replica to its replica cache and sends a broadcast message *RPCA_ADD_REPLY* to inform all nodes that a new replica is added. Finally, node *k* changes its state from *Added* to *Start* when it receives *RPCA_ADD_REPLY*.

Second, replica node *k* decides whether to delete the replica from its local cache. If node *k* satisfies condition (3.2), it changes its state from *Start* to *Deleting*. Even if node *k* satisfies (3.2), it cannot delete the replica immediately because there may be several replica nodes in the system that satisfy (3.2), so deleting without the necessary coordination may cause the number of replicas in the system to drop to zero. Replica node *k* in the *Deleting* state sends coordination message *RPCA_DELETING* to other

replica nodes to request permission to delete its replica. Replica nodes send a permission message back to node *k* only when its state is *Deleting*. We denote the number of permission messages that replica node k receives as *DRP* and the current number of replica as *CRP*. If *CRP - MIN > DRP*, which means replica node *k* has enough permissions to delete its replica, replica node *k* changes its state from *Deleting* to *Deleted*. Otherwise, it changes its state from *Deleting* to *Start*. When a replica node *k* is in *Deleted* state, it deletes its replica and sends message *RPCA_DELETE* to inform all nodes in the system that its replica is deleted. Finally, it changes its state from *Deleted* to *Start*.

Third, if both attempts to add and delete a replica fail and *F* reaches *MAX*, *FDRM* applies the migration process. Replica node *k* scans the *outRead* list to find the first node *j* ($j \neq k$) that satisfies (3.4) and changes its state from *Start* to *Switch*.

$$R_{out,j} > R_{in} \qquad (3.4)$$

When replica node *k* is in the *Switch* state, it sends a message *RPCA_SWITCH* to node *j* and deletes its local replica. Upon receiving *RPCA_SWITCH* from replica node *k*, node *j* copies the replica from message *RPCA_SWITCH* to its local cache and sends a broadcast message *RPCA_SWITCH_REPLY* to all nodes in the system to indicate the switching of replica from node *k* to node *j*. Finally, when node k receives *RPCA_SWITCH_REPLY*, it changes its state from *Switch* to *Start*.

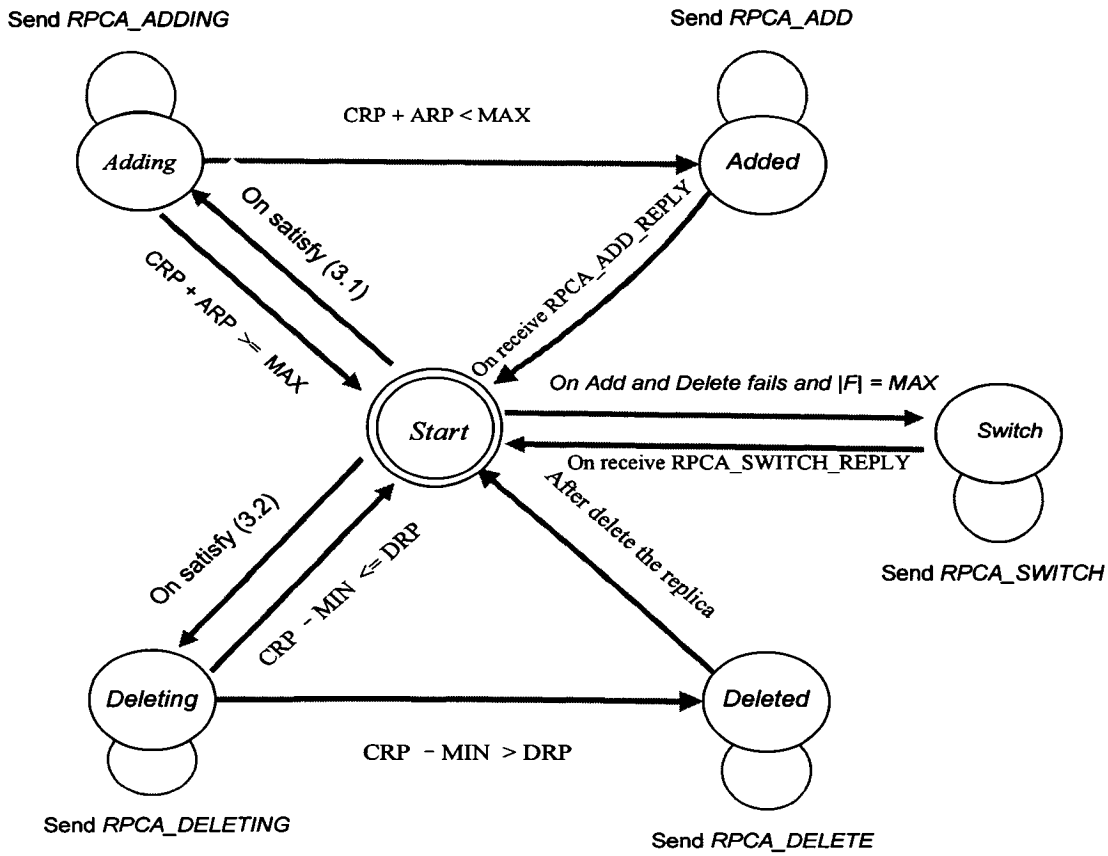

Figure 3.1 States of replica allocation phase in FDRM

3.3 The ARAM Algorithm

ARAM is a dynamic replication algorithm proposed to minimize the communication cost of data access and adapt to node mobility. The major difference between ARAM and FDRM is that replicas are added or switched to neighboring nodes. The replica allocation in ARAM is based not only on local replica access information but also on replica access information from other nodes. Reads are satisfied from a local cache or from the closest replica node *k*. Writes are first sent to the closest replica node *k*, which forwards writes to other replica nodes. The notation used in the ARAM algorithm is defined in Table 3-2.

Notation	Definition
F	the set of replica nodes in the system
d(i, j)	the shortest hop distance between node *i* and node *j*
q(i)	the weighted value of a replica node *i*, *q(i)* = $\sum_{j \in F} d(i,j)$
C (j)	the access set of a non-replica node *j*, *C (j)* = {*i* / $\min_{i \in F}$ *d (j, i)*}
d(j, F)	the cost of a read request from a non-replica node *j*. If there are multiple nodes which all have the fewest hop distance to node j, node j access the replica set from the replica node which has the least weight value. *d(j, F) = d(j, k)*, *q(k) = min (q(m)) and m, k* $\in C(j)$
R(i)	the number of reads sent by node *i* during a fixed time interval t
W(i)	the number of writes sent by node *i* during a fixed time interval t
P(i)	the number of writes that a replica node i receives from itself and its non-replica neighbors.
Wf(V)	the cost of forwarding writes to the closest replica nodes *Wf(V)* = $\sum_{i \in V} W(i)d(i,F)$
Wp (V)	the cost of propagating writes, *Wp(V)* = $\sum_{s \in F} \sum_{j \in F} P(s)d(s,j)$
R(V)	The read cost of all nodes, *R(V)* = $\sum_{i \in V} R(i)d(i,F)$

Table 3-2 Notation in ARAM

3.3.1 The Cost Model

ARAM assumes each data access is independent. The costs of a read request from a node *j* are defined as *d (j, F)*. The costs of a write request from a node *j* are defined as *d (j, F)* + $\sum_{k \in F} d(s,k)$, and node *j* accesses the replica set *F* through node *s*.

Figure 3.2 gives an illustration of replica access in ARAM.

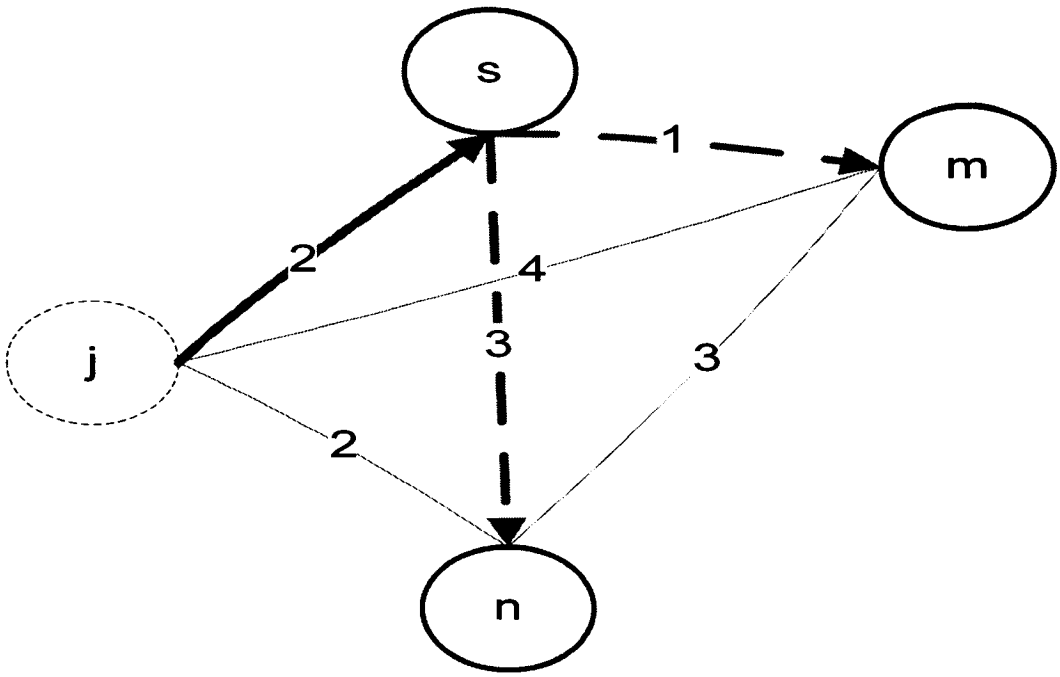

Figure 3.2 Replica access in ARAM

In Figure 3.2, *j* is a non-replica node and *s, m* and *n* are replica nodes. The numeric number denotes the shortest hop distance between two nodes. The solid bold line represents a write request from node *j* is first sent to a replica node *s* while two dash lines represents write requests are forwarded to replica node *n* and replica node *m*. Based on the definitions in Table 3-2, we have the following equations:

- F = {s, m, n) and V = {j, s, m, n}
- d(j, F) = 2 and C(j) = { s, n}
- q(s) = d(s, m) + d(s, n) = 1 + 3 = 4
- q(n) = d(n, m) + d(n, s) = 3 + 3 = 6.

Because q(s) is less than q(n), node *j* accesses *F* through node *s*. A write request is first forwarded to node *s*, which propagates the write request to node *n* and node *m*. The total communication cost for all nodes during a time interval t is defined in (3.5).

$$Cost(V) = Wf(V) + Wp(V) + R(V)$$

$$= \sum_{j \in V} W(j)d(j,F) + \sum_{s \in F} \sum_{k \in F} P(s)d(s,k) + \sum_{j \in V} R(j)d(j,F) \qquad (3.5)$$

3.3.2 Replica allocation in ARAM

Table 3-3 shows the definitions of variables which are used to add replicas in ARAM. In Figure 3.3, non-replica node *u* is a neighbor of replica node *s*. If a replica is added to *u*, one hop is decreased for some nodes to access the replica. The write cost, however, increases for propagating writes to the new replica node *u*. Let *V* be the set of all nodes in the system. The condition of replica addition is expressed in (3.6).

V	all mobile nodes in the system
F	replica nodes in the system
u	a neighboring node of replica node s, u $\in$ *V - F, s* $\in$ *F, d(s, u) = 1*
A	*A = { i / i* $\in$ *V - F, s* $\in$ *C (i)* and *u* is on one of the shortest paths between *i* and *s*, and i access F through p_i }
R(u)	the number of reads received by *s* from *u*
W(u)	the number of writes received by *s* from *u*
R(A)	the number of reads from set *A*
W(A)	the number of writes from set *A*
WT(k)	the number of writes that replica node *k* receives from other nodes and itself
WA (k)	the number of writes that replica node *k* received from set *A*
W	the number of writes from set *V*
$\Delta change$	the change of write cost caused by the change of the access path of *A*. $\Delta change = \sum_{i \in A} W(i)(q(s) - q(Pi))$

Table 3-3 Variables used to add replicas in ARAM

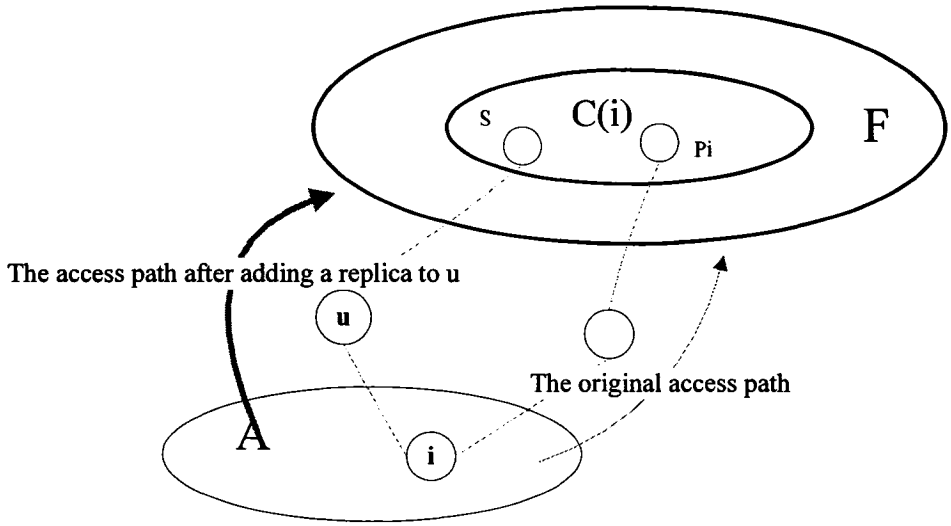

Figure 3.3 An illustration of adding a replica in ARAM

$$R(u) + R(A) > (|F| - 2)(W(u) + W(A)) + \sum_{k \in F}(WT(k) - WA(k))d(k,s) + W + \Delta change \quad (3.6)$$

Table 3-4 shows variables used in deleting and switching replicas. In Figure 3.4, if the number of writes received by a replica node *s* from other replicas is larger than the reads and writes received by *s* from itself and non-replica nodes, then *s* deletes the replica. The condition of replica deletion is expressed in (3.7).

Notation	Definition
B	*B* $\{i \mid i \in V - F$, *i* access F through *s and* $C(i) - \{s\} \neq \phi\}$
R(B)	the number of reads from set *B*
W(B)	the number of writes from set *B*
RT(s)	the number of reads received by *s* from itself and other nodes
WT(s)	the number of writes received by *s* from itself and other nodes
p_i'	for $i \in B$, $p_i' \in C(i)$ *and* $q(p_i') = min\ q(j)$, $j \in C(i)-\{s\}$.
z	$z \in F - \{s\}$ *and* $d(s, F - \{s\}) = d(s, z)$
w	$w \in F - \{s\}$, $q(w) - d(w, s) = max(q(k) - d(k, s))$, $k \in F - \{s\}$
ΔQ	$\Delta Q = \max(1, q(s) + \|F\| - 1 - q(j))$, $j \in F - \{s\}$

Table 3-4 Variables used to delete and switch replicas in ARAM

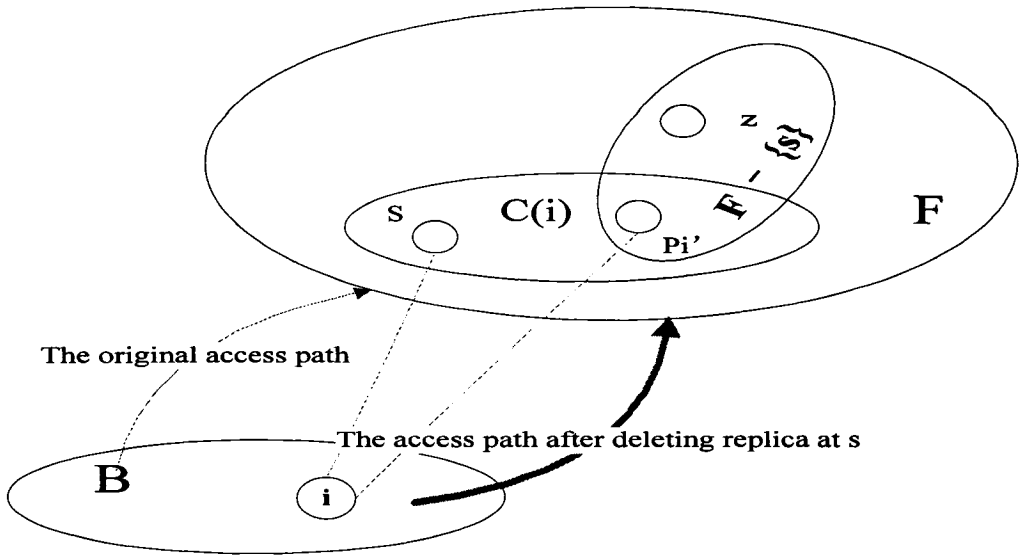

Figure 3.4 An illustration of deleting a replica in ARAM

$$(R(B) + W(B))\, d(s, z) + \sum_{i \in F} WT(i) d(i, s) > (RT(s) + WT(s))\, d(s, z) + WT(s)\,(q(w) - q(s) - d(s, w)) + W(B)(q(p_i') - q(w) - d(s, p_i') + d(s, w)) \quad (3.7)$$

A replica node *s* switches a replica to a neighbor node *u* when *u* receives more reads and writes than replica node *s*. The condition of replica switch is expressed in (3.8).

$$R(u) + W(u) > 1/2\,(RT(s) + WT(s) + \sum_{j \in F - \{s\}} WT(j) \Delta Q) \quad (3.8)$$

3.3.3 Details of ARAM

Appendix B shows the details and pseudo code of ARAM. The ARAM algorithm does not give detailed information about how to collect reads and writes from other nodes and how to calculate the replica allocation intervals. In our simulation, we adopt the same allocation interval scheme in FDRM. We use the blind flooding algorithm [27] to collect data access information from other nodes. When replica node *s* starts the replica allocation process, *s* sends a broadcast message *ALLOCATE* to inform all nodes in the network that it is starting to collect data access information. Upon receiving the

ALLOCATE message, node *i* replies with the total number of read and write requests that node *i* sent during the time interval t. If *i* is a replica node, it also replies with the total number of read and write requests that node *i* received during the time interval *t*.

3.4 The MFDRM Algorithm

MFDRM is an extension of FDRM and ARAM that works in MANETs. In MANETs, the communication cost between two nodes includes the wireless bandwidth cost, energy consumption, and the delay of communication. Since all those factors are directly related to the number of hops on a communication path, we use the number of hops between two nodes to measure the communication cost between these two nodes. FDRM assumes that the system cost to satisfy a read request, α , and the system cost to update a replica, β , are the same. This assumption is not accurate in MANETs because of the multi-hop nature. Replica allocation in MFDRM considers both the data access load and the hop distance for each data access, which is more suitable in MANETs. Appendix C has detailed pseudo code of algorithm MFDRM.

3.4.1 The Cost Model

Table 3-5 shows the notation used in MFDRM. We consider a general network consisting of a set of mobile nodes $V = \{1, 2 \ldots n\}$. F is a set of replica nodes and $F \subseteq V$. A non-replica node *j* may access a replica at a replica node *i*. The cost of such an access is denoted as *d(i, j)*, where $d(i, j) \geq 0$ and *d(i, j)* is the shortest number of hops between node *i* and node *j*. The cost of a read request from mobile node *i* is defined as $\min_{j \in F} d(i, j)$. The cost of a write request from mobile node *i* is defined as $\sum_{j \in F} d(i, j)$. If we assume the number of replicas over a time duration t is constant, the total communication cost of all nodes during t can be expressed in (3.9).

Notation	Definition
R(i)	the number of reads from node *i*
W(i)	the number of writes from node *i* during a time period t
F	all replica nodes in the system
V	all nodes in the system
d(i, j)	the shortest number of hops between node *i* and node *j*
RL	a list whose entry represents the number of reads that node *i* receives. For example, *RL(j)* means the number of reads that are sent from node *j* and received by node *i*
WL	a list whose entry represents the number of writes that node *i* receives. For example, *WL(k)* means the number of writes that are sent from node *k* and received by node *i*.

Table 3-5 Notations in MFDRM

$$\text{Cost(V)} = \sum_{i \in V} R(i)(\min_{j \in F} d(i,j)) + \sum_{i \in V} \sum_{j \in F} W(i)d(i,j) \qquad (3.9)$$

3.4.2 Replica allocation in MFDRM

We assume the read-write pattern during a time period is in most cases predictable based on the read-write pattern in the preceding time period. After adding a replica to a non-replica node *j*, the read cost decreases because reads from node *j* can be satisfied from its local cache. On the other hand, the write cost increases since writes from other nodes must be propagated to node *j*. The criterion for a replica node *i* to add a replica to a non-replica node *j* which reads from it, is that the overall decreases of read costs are greater than the overall increases of write costs. The condition of replica addition is expressed in (3.10).

$$RL(j) \bullet d(i,j) > \sum_{k \in WL} (WL(k) \bullet d(k,j)) + W(i) \bullet d(i,j) \qquad (3.10)$$

If a replica is deleted from node *i*, read costs increase because node *i* must read a replica from another replica node *m*. On the other hand, write costs decrease since writes need update fewer replicas. The criterion that a replica node *i* delete its replica, is that the overall increase in read costs is less than the overall decrease in write costs. The condition of replica deletion is expressed in 3.11

$$R(i) \bullet d(i, m) + \sum_{j \in RL} RL(j) \bullet (d(j,q) - d(i,j)) < \sum_{k \in WL} WL(k) \bullet d(i,k) \qquad (3.11)$$

$$m = \min_{p \in F'} d(i, p) \qquad q = \min_{p \in F'} d(j, p) \qquad F' = F - \{i\}$$

m denotes the replica node from which node *i* reads a replica after node *i* deletes its replica. *q* denotes the replica node from which non-replica node *j* reads a replica after node *i* deletes its replica.

3.4.3 An illustration of MFDRM

In this section, we give an illustration of MFDRM. In Figure 3.5, there are five mobile nodes M_1, M_2, M_3, M_4 and M_5. M_1 and M_5 have replicated data *D*. The numeric number of an edge denotes the hop distance between two nodes. For example, $d(M_1, M_2) = 1$ and $d(M_3, M_5) = 2$. The direction of an edge means a non-replica node reads a replica from a replica node. For example, M_2 and M_3 read replica *D* from M_1 while M_4 reads replica *D* from M_5.

Table 3-6 shows the data access load, hop distance for reads and writes, and communication costs for each mobile node. The communication cost is defined as the number of data accesses times the hop distance. For example, M_2 has 100 reads and 10 writes. M_2 reads replica *D* from node M_1 with a hop distance $d(M_1, M_2) = 1$. Therefore, the read cost for M_2 is 100 * 1 = 100. For a write access, M_2 must update two replicas in M_1 and M_5. The overall hops distance is $d(M_1, M_2) + d(M_2, M_5) = 3$. Therefore, the write cost for M_2 is 10 * 3 = 30. The total read cost for all nodes is 140 and the total write cost for all nodes is 146.

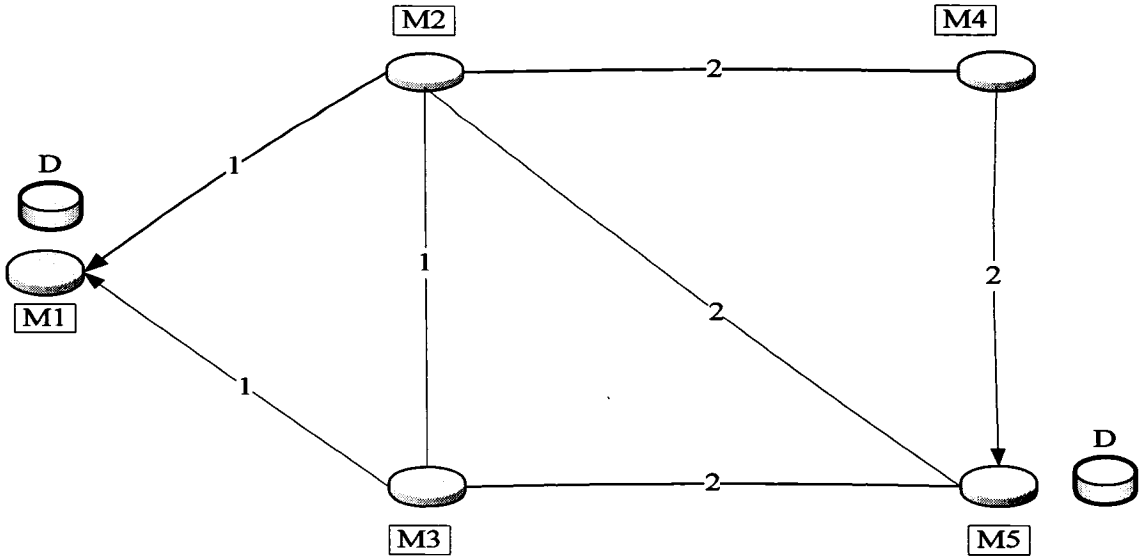

Figure 3.5 Replica placements in MFDRM before allocation

Hosts	Reads			Writes		
	load	hops	Access cost	load	hops	Access cost
M_1	40	0	0	30	3	90
M_2	100	1	100	10	3	30
M_3	20	1	20	0	3	0
M_4	10	2	20	0	5	0
M_5	30	0	0	8	3	24
Total			140			146

Table 3-6 Data access cost before replica allocation in MFDRM

At the end of the replica access phase, the replica node M_1 enters the replica allocation phase. Based on its local data access information, M_1 tries to find a non-replica node which reads replicas from it and satisfies the condition of replica addition. In this example, M_1first tries to add a replica $D$to M_2. The read cost decrease for $M_2$is 100 because the reads can be satisfied from its local cache. The write cost increases include

the write cost increase from M_1, which is 30, and the write cost increase from M_5, which is 16. The overall write cost increase, therefore, is 46. Based on (3.10), the condition for a replica to be added to M2 is satisfied. For M_3, the read cost decrease is 20. The write cost increases includes the write cost increase from M_1, which is 30, and the write cost increase from M_5, which is 16, and the write cost increase from M_2, which is 30. M_3 does not satisfy condition (3.10). Thus, a replica D is added to node M_2. Table 3-6 shows that the read cost decreases from 140 to 40 while the write cost increases from 146 to 190. However, the total communication cost decreases from 286 to 230.

Hosts	Read			Write		
	load	Hops	access cost	Load	hops	Cost
M_1	40	0	0	30	4	120
M_2	100	0	0	10	3	30
M_3	20	1	20	0	4	0
M_4	10	2	20	0	7	0
M_5	30	0	0	8	5	40
Total			40			190

Table 3-7 Data access cost in MFDRM after adding replicas

After the addition, M_1 decides whether to delete the replica of D from itself. The overall read cost increases include the read cost increase from node M_1, which is 40, and the read cost increase from node M_3, which is 0 because $d(M_3, M_1)$ equals $d(M_3, M_2)$. The overall read cost increase, therefore, is 40. The overall write cost decreases include the write cost decrease from node M_2, which is 10, and the write cost decrease from M_5, which is 24. The overall write cost decrease is 34. Based on the inequality 3.11, the deletion is not satisfied.

3.5 Summary

In this chapter, we present a detailed description of the selected dynamic replication algorithms. Section 3.1 outlines the assumptions and definitions of the selected dynamic replication algorithms. Section 3.2 and Section 3.3 discuss the details of the FDRM and ARAM algorithms. Section 3.4 discusses MFDRM, which is our extension of FDRM and ARAM.

There are five major differences between these algorithms. First, replica allocation in ARAM includes both local and remote read/write information, which makes the algorithm complex. On the other hand, replica allocation in FDRM and MFDRM only includes local read/write information. Second, the replica allocation in ARAM and MFDRM consider both the workload in each replica node and the hop distance of each read/write request, which makes the algorithms more accurate in MANETs. Replication allocation in FDRM considers only the workload in each replica node. Third, adding and switching replicas in FDRM and MFDRM applies to any node in the system while adding and switching replicas in ARAM only applies to neighboring nodes. Fourth, both FDRM and MFDRM control the number of replicas in the system, which guarantees that the number of replicas in the system does not become zero. ARAM, however, does not have any mechanism to control the number of replicas in the system. Fifth, the replica update procedure in ARAM is different from the replica update procedures in FDRM and MFDRM. Each write request in ARAM is first forwarded to the closest replica node, which has the least weight value, and this node forwards replica update requests to other replica nodes in the system. Each write request in FDRM and MFDRM is sent to all replica nodes in the system individually. In Chapter 4, experimental results are presented and analyzed in terms of selected performance metrics under different simulation

scenarios.

Chapter 4

Performance Evaluation

In this chapter, we compare the dynamic replication algorithms presented in Chapter 3 to a static replication algorithm, which we denote as SDRM. Each read request in SDRM is sent to the closest node while each write request is sent to update all replicas. The number of replicas in SDRM does not change during the simulation. Various tests are done to demonstrate trade-offs between efficiency, availability, and consistency. Based on our simulation results, we provide suggestions for improving certain aspects of dynamic replication systems that are suitable for MANETs.

Section 4.1 introduces the simulation model. Section 4.2 discusses the performance metrics, factors that affect the performance and simulation parameters. Section 4.3 discusses the simulation results and analyzes the performance of the selected replication algorithms under different scenarios. Section 4.4 summarizes this chapter.

4.1 Simulation Model

The Network Simulator Version 2, namely NS-2 [29], is used in our simulation experiments. NS-2 is an object-oriented and discrete event-driven network simulator, which provides support for wired and wireless networks. NS-2 provides substantial support for the simulation of TCP, routing, and multicast protocols over wired and wireless networks. NS-2 separates detailed protocol implementation from simulation configuration and is, therefore, the most popular network simulator for MANETs. All algorithms are simulated under the same settings to guarantee the fairness of the simulation results. All results presented are the average of ten simulation runs.

Our simulation model consists of four models: (1) Mobility Model, (2) Workload Mode, (3) Energy Model and (4) Network Model.

4.1.1 Mobility Model

The mobility patterns [33] refer to how a mobile node moves in a network area. Given a fixed radio communication range for mobile nodes, mobility is the main reason for network partitioning and dynamic topology. The mobility pattern affects the replica availability and consistency. The random waypoint mobility model [30] is used in our simulation. In this model, the starting and ending positions of a node movement, called waypoints, are uniformly distributed inside the simulation area. Motion is characterized by two factors: (a) a maximum speed and (b) a pause time. During simulation each node starts moving from one waypoint to another waypoint in a straight line with constant velocity. The velocities for different nodes are uniformly distributed between 0 and the maximum speed. When a node reaches the target waypoint, it waits for the pause time. Afterwards, it proceeds by selecting another random target waypoint and moves to it and so on. In our simulation, we choose the maximum node movement speed as 5 m/s.

4.1.2 Energy Model

Mobile devices are usually battery-powered and one of the design objectives for a mobile system is minimizing power consumption in order to increase the lifetime of mobile devices. NS-2 provides a simple energy model, called EnergyModel, to model the energy consumption of a mobile node. Parameters, such as initial energy, packet sending energy and packet receiving energy, must be specified for this model. In our simulation, each mobile node has the same initial energy of 500 joules, which guarantees all mobile nodes are active during the simulation. Sending and receiving each packet consumes 0.6w and 0.4w of energy, respectively. Although in practice

mobile nodes also consume energy while listening, we assume for simplicity that the listen operation for each mobile node is energy free.

4.1.3 Workload Model

Workload has a significant impact on the performance of replication systems. Since no real workloads have been collected from MANETs, we vary the read-write ratio and the packet sending rate to emulate the changes in workload. A higher read-write ratio generates less traffic than a low read-write ratio because writes update all replicas while reads only access one replica. A higher packet sending rate generates more message traffic than a lower packet sending rate. The traffic source in our simulation is a constant bit rate application. In our simulation, we choose 20 nodes as traffic sources. Each packet sent by the source has a fixed size payload of 2KB.

The simulation takes 900 seconds [29] and consists of two phases. The length of the first phase is 200 seconds. Ten mobile nodes in the system are selected randomly as replica nodes at the beginning of simulation. Different replica nodes send a broadcast message at different time units in the first phase to indicate to other nodes that they have the replicas. This guarantees each mobile node has a complete list of replicas before the actual data access phase starts. The second phase is 700 seconds long. Each node generates reads and writes based on the simulation settings. The time for each node to start sending reads and writes is randomly distributed among the 700 seconds. A node keeps sending reads and writes until the simulation ends.

4.1.4 Network Model

Each mobile node in the ad-hoc network has the same initial value settings, such as energy, memory and computation power. All mobile nodes are distributed randomly inside a terrain of size 1000 meters * 1000 meters. All simulated algorithms are

implemented in the application layer. 802.11 [31] is the MAC layer protocol. The bandwidth is 2Mb/s. The Ad-hoc On-Demand Distance Vector (AODV) routing protocol [32] is used as the unicast routing protocol. Broadcast messages are sent using the blind flooding [27] algorithm.

4.2 Performance Metrics and Factors

Five performance metrics, namely read response time, read fail ratio, write propagation delay, write fail ratio and energy consumption, are used to analyze the performance of the four algorithms. Four important factors read-write ratio, pause time, packet sending rate and network size, are varied over a certain ranges to examine their impact on the performance of the algorithms.

4.2.1 Performance Metrics

1. **Read response time**

   Read response time is the time that elapses from the time a mobile node sends a read request until it receives an object. Read response time is assumed to be zero when a node has a replica stored locally. The average of all read response times during the length of the simulation is calculated. A fundamental goal of data replication in distributed systems is to ensure that read response time is low. In a MANET, response time becomes the limiting factor in data access due to the limited bandwidth and processing capabilities. The majority of replicated data in real scenarios is read-only and the read response time is an indicator of the efficiency of data access in the system.

2. **Read fail ratio**

   Read fail ratio is defined as the percentage of failed read requests among all read requests during the simulation. In wired networks, all nodes are strongly connected with each other and the read fail ratio is zero in the absence of node and or link

failure. In MANETs, the read fail ratio would normally be higher than the read fail ratio on wired network because the network can be partitioned at any time. The probabilities of node failure and link interference in MANETs are high as well. A low read fail ratio implies high data availability. This metric is an indicator of the data availability.

3. **Write propagation delay**

   Write propagation delay is defined as the time that elapses from the time a mobile node sends a write request until all replicas in the system have been updated. There are trade-offs between low read response time and high write propagation delay. An algorithm may have a low read response time but a high write propagation delay. A low write propagation delay also reduces the chances of replica update conflicts.

4. **Write fail ratio**

   Write fail ratio is defined as the percentage of failed write requests among all write requests during the simulation. A write request succeeds only when all replicas in the network are updated. This metric is an indicator of replica consistency. A low write fail ratio indicates a high percentage of replicas in the system have the most recent data.

5. **Energy consumption**

   The average energy consumption of all nodes in the network is computed. We assume that all mobile nodes are active during the simulation and mobile nodes consume energy only when it sends or receives packets. In this way, the average energy consumption of all mobile nodes during the simulation is proportional to the received and sent packets during the simulation. An algorithm that has low energy consumption improves the lifetime of the network.

4.2.2 Factors affecting Performance

1. **Read-write ratio**

   The read-write ratio is defined as the number of read requests versus the number of write requests during the simulation. Dynamic replication expands or shrinks the number of replicas in the system based on the number of read and write requests in the network. Varying this factor affects the number of replicas in a dynamic replication system.

2. **Pause time**

   Pause time is defined as the time period during which a node is stationary after arriving at a destination. A higher pause time results in lower mobility. Varying this factor demonstrates the ability of the algorithms to deal with node mobility. The ability to deal with node mobility is one of the primary goals for any algorithm in MANETs.

3. **Packet sending rate**

   Packet sending rate is defined as the number of requests sent within a second for each node that generates workload. A high packet sending rate implies the MANET has a high workload. Increasing the workload may cause undesirable changes in network performance, such as high latency, packet dropping and control overhead. We observe the performance of the algorithms under different workloads by varying this value.

4. **Network size**

   Network size is defined as the number of nodes inside the simulation area. It is an indicator of node density in the system. A high node density implies a low chance of network partition. We observe algorithms under different nodal densities by varying this value.

4.3 Simulation Results and Analysis

In this section we evaluate the performance of SDRM, FDRM, ARAM and MFDRM. Multiple simulation scenarios are designed to observe the efficiency, availability, consistency and energy consumption. Table 4-1 lists the fixed parameters of our simulation model, which are discussed in section 4.1.

Parameters	Values
Number of replicas at the beginning	10
Number of nodes sending packets	20
Maximum movement speed	5 meter / second
Simulation time	900 seconds
Initial energy	500 Joules
Packet size	2KB
Bandwidth	2M bit / second
Sending energy	0.6 w
Receiving energy	0.4 w
Simulation area	1000 meter * 1000 meter
Routing protocol	AODV
Broadcasting protocol	Blind flooding
Minimum number of replica	1
Maximum number of replica	25
Minimum allocation interval	20 seconds
Maximum allocation interval	200 seconds
Allocation interval at the beginning	60 seconds

Table 4-1 simulation environment parameters with fixed values

4.3.1 The effects of read-write ratio

Table 4-2 lists the parameter settings in this experiment. Read-write ratio is varied from 1:1 to 40:1 to simulate scenarios from a high percentage of writes to a high percentage of reads.

Figure 4.1 shows the difference between the initial number of replicas and the number of replicas after the simulation. This reflects whether the replica set expands or shrinks during the simulation. Figures 4.2 to 4.6 show the simulation results of all algorithms under different read-write ratio scenarios.

Parameters	Value
Read-write ratio	1:1, 10:1, 20:1 and 40:1
Transmission range	230 meters
Number of nodes in the system	30
Number of nodes sending packets	20
Number of replica nodes at the beginning	10
Packet sending rate	1 packet/2 seconds
Pause time	50 seconds

Table 4-2 experiment parameters in read-write ratio

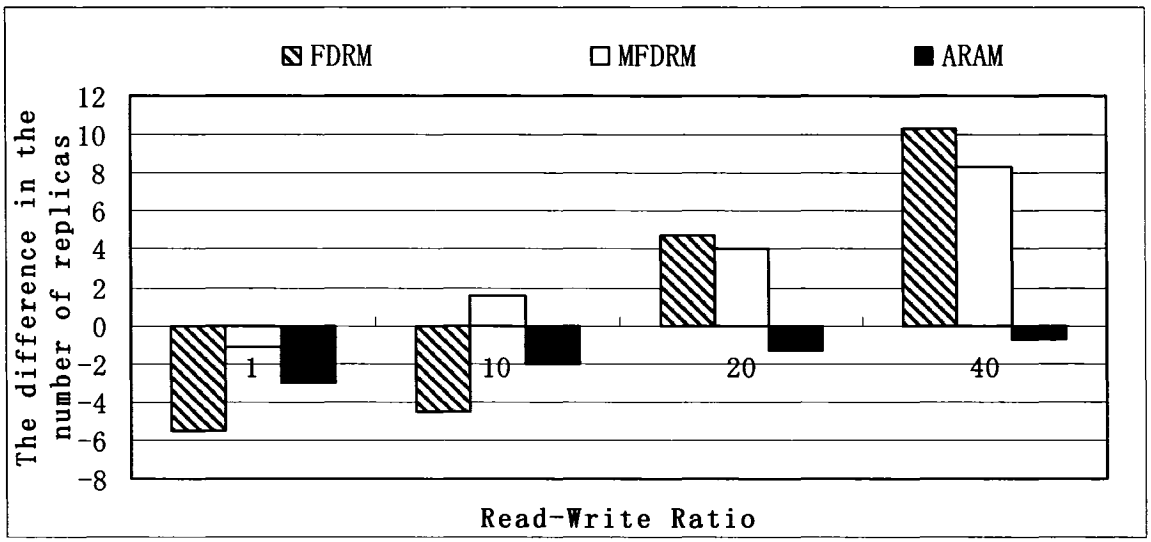

Figure 4.1 Effects of read-write ratio on the difference in the number of replicas

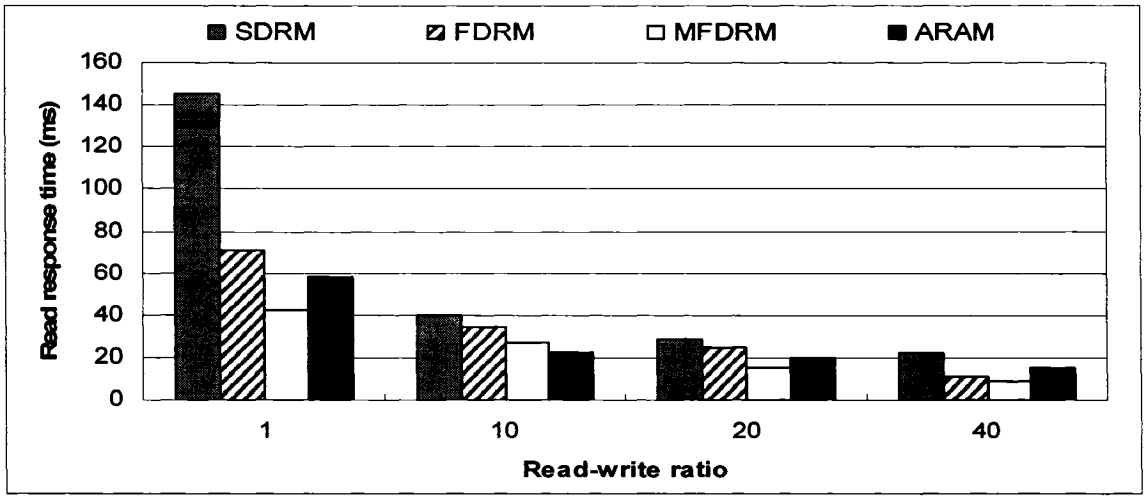

Figure 4.2 Effects of read-write ratio on read response time

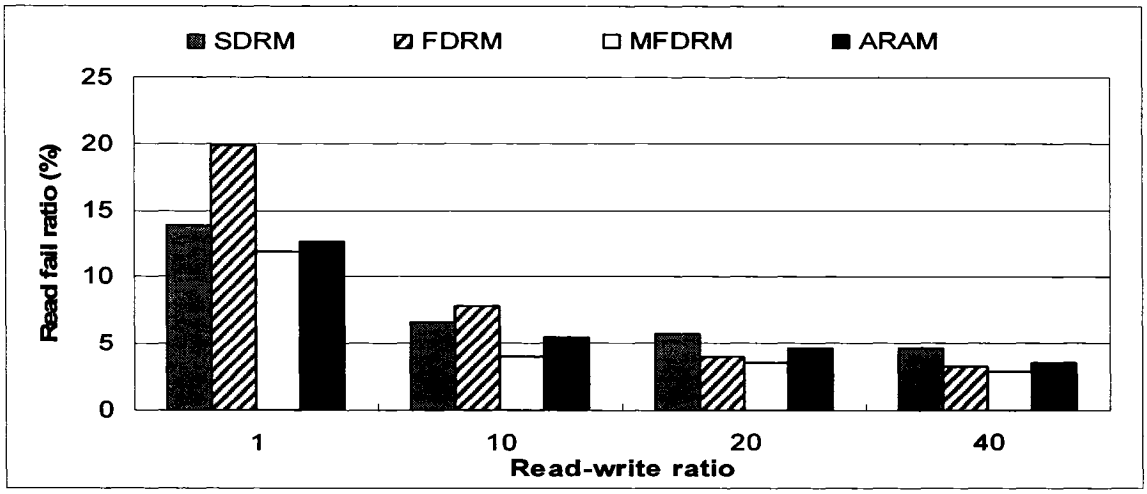

Figure 4.3 Effects of read-write ratio on read fail ratio

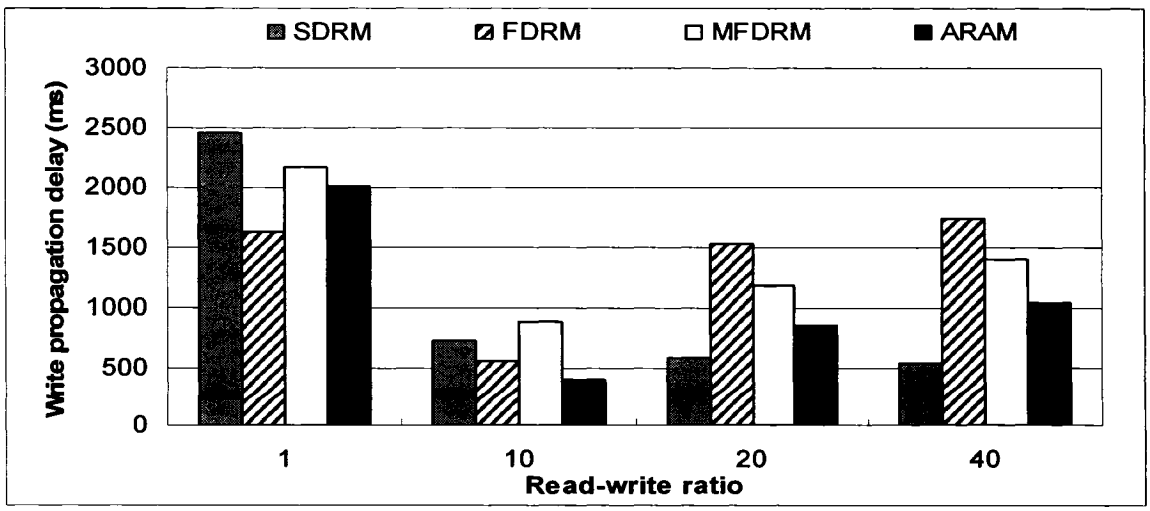

Figure 4.4 Effects of read write-ratio on write propagation delay

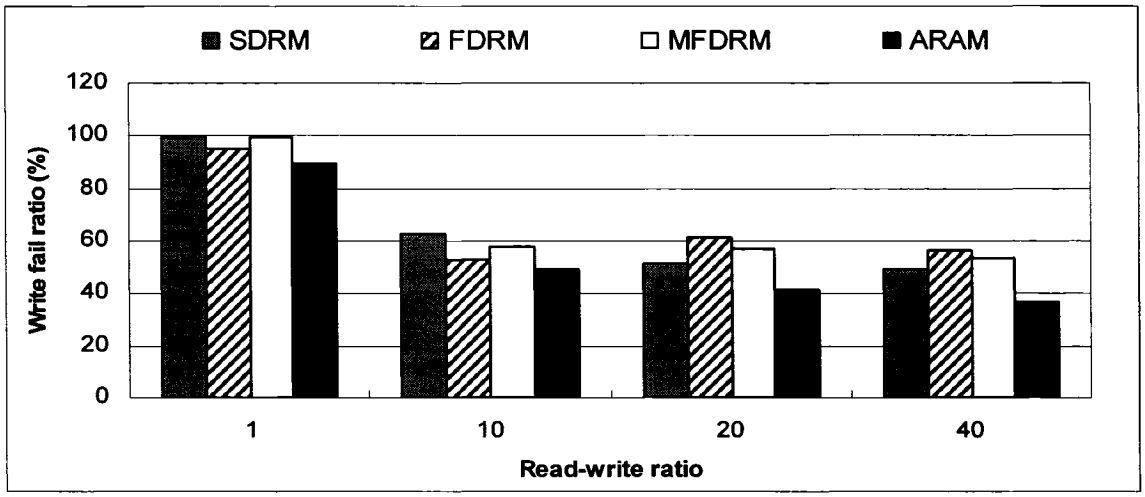

Figure 4.5 Effects of read-write ratio on write fail ratio

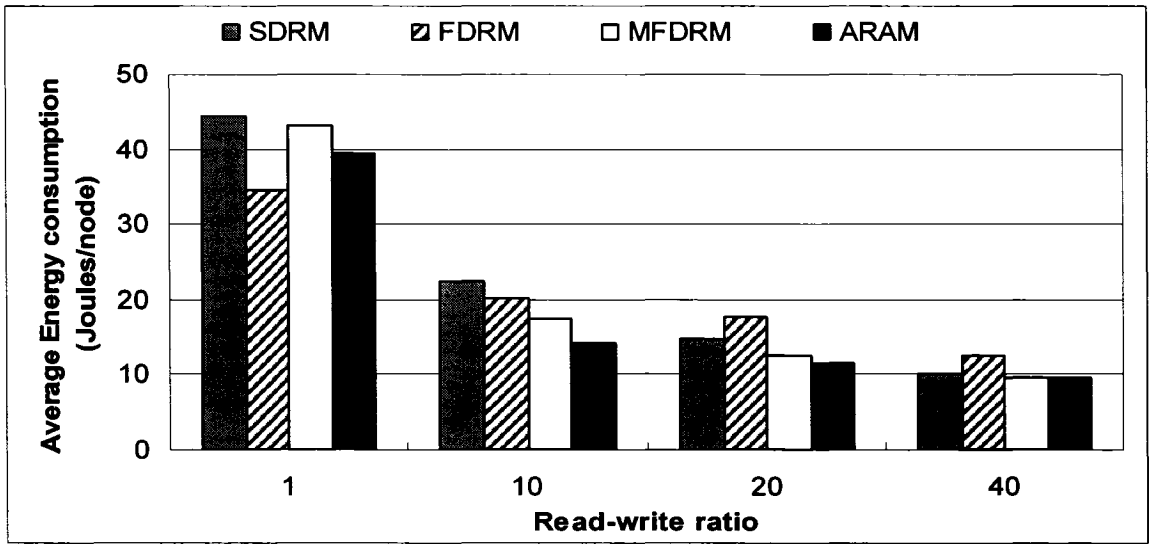

Figure 4.6 Effects of read-write ratio on average energy consumption

From Figure 4.1, we see that the replica sets of ARAM, FDRM and MFDRM shrink when the read-write ratio is set to 1. The reason is that writes are sent to all replicas in the network while reads are satisfied from the local cache or the closest replica node. There are more write packets than read packets during the simulation when the read-write ratio is set to 1. For example, if every write packet is delivered to a destination, a replica node *k* may receive 20 writes since the number of traffic source

nodes is 20. Meanwhile, the read requests that a replica node *k* receives are most likely less than 20. Since the replica nodes and the traffic source nodes are distributed uniformly in the system, the chances that all nodes read a replica from a single replica node *k* is very low. The replica set of dynamic replication algorithms shrink when a replica node receives more writes than reads. Replica allocation in FDRM only considers the workload while replica allocation in ARAM and MFDRM considers both workload and hop distance for each data access. The consequence is that the number of replicas in FDRM shrinks significantly when the workload has a high percentage of writes while the number of replicas in FDRM expands significantly when the workload has a high percentage of reads.

The number of replicas in ARAM does not expand when the workload has more reads than writes. This demonstrates that the replica adding condition in ARAM is very restricted and does not respond well as the number of reads increase. On the other hand, the number of replicas in MFDRM only has a slight decrease when the workload has a higher percentage of writes. From (3.11) in MFDRM, we see the read cost increase, caused by deleting its replica, contains both the read cost increases from a replica node *k* and the read cost increases from other non-replica nodes that read from replica node k. From (3.2) in FDRM, we see the read cost increase, caused by deleting its replica, only contains the read cost increase from replica node *k* itself. Deleting replicas in MFDRM, therefore, is more restricted than deleting replicas in FDRM. MFDRM therefore does not respond well as the number of writes increases.

The read response time and read fail ratio in Figure 4.2 and Figure 4.3 decrease as the read-write ratio increases. There are two reasons. First, the number of packets sent during the simulation decreases as the read-write ratio increases because writes

generate more packet transmissions than reads. As fewer writes are sent, the workload decreases. This results in the decrease of each packet's end-to-end delay. Second, the number of replicas created increases or equivalently, the number of replicas deleted decreases during the simulation as the read-write ratio increases.

Simulation results in Figure 4.2 show that SDRM has a higher read response time than dynamic replication algorithms when the workload has a high percentage of write requests, such as when the read-write ratio is set to 1. The reason is that the number of replicas and their locations in SDRM do not change during the simulation. When the read-write ratio is set to 1, FDRM has a higher read response time than ARAM and MFDRM because it has the lowest number of replicas among all algorithms. As the read-write ratio increases, the replica set of FDRM expands and its read response time decreases as well. We also notice that when FDRM has more replicas than MFDRM, its read response time is higher than MFDRM. This demonstrates that MFDRM, which distributes the data based on both the workload and hop distance of each data access, outperforms FDRM, which distributes replicas based only on workload.

From Figure 4.3, we see FDRM has the highest read fail ratio when the workload has a high percentage of write requests such as a read-write ratio of 1. The reason is that its replica set shrinks from 10 to 4.5. As the read-write ratio increases, the replica set of FDRM increases which results in its read fail ratio decreasing. ARAM has a lower read fail ratio than the static replication algorithm SDRM although ARAM has fewer replicas than SDRM.

The write propagation delay in Figure 4.4 decreases when read-write ratio increases from 1 to 10. The reason is that the packet end-to-end delay is reduced as

fewer writes are sent during the simulation. When the read-write ratio is 1, SDRM has the highest write propagation delay because it has largest number of replicas among all algorithms. FDRM has the lowest write propagation delay because it has the lowest number of replicas. When the read-write ratio increases from 10 to 40, the write propagation delays of FDRM, MFDRM and ARAM increase while the write propagation delay of SDRM decreases slightly. The reason is that as the read-write ratio increases, the number of created replicas increases or the number of deleted replicas decreases which means more replicas need to be updated for algorithms ARAM, FDRM and MFDRM. The replica set in SDRM does not change and the workload is reduced as fewer writes are sent. SDRM therefore has a slight decrease in write propagation delay.

The write-fail ratio in Figure 4.5 is fairly high (90% - 99%) when the read-write ratio is 1. There are three reasons for such a situation. First, writes which are sent to update all replicas in the system generate large amounts workload. As the workload increases, the packet delivery ratio decreases. Second, no reliable multicast protocol is used. Control messages, which indicate the changes in replica set information, may not be delivered to all replica nodes. This results in an inconsistency between the local view of the replica set and the global view of the replica set. Third, since mobile nodes move freely in MANETs, network partitioning is frequent which makes updating all replica nodes impossible. This also leads us to suggest that the replica consistency protocol, Read-One-Write-All, is not suitable for MANETs when the network workload has a high percentage of writes.

From Figure 4.5, we see ARAM has the lowest write fail ratio among all algorithms. The reason is that a write request is first forwarded to the closest replica node that has the least weight value. This approach minimizes the total hop distance of the replica

update path. A replica update path that has fewer hops tends to reduce the write fail ratio.

The average energy consumption in Figure 4.6 reflects the total number of packets sent and received during the simulation. Since the total number of packets sent and received during the simulation decreases as the read-write ratio increases, the average energy consumption of each node decreases for all algorithms. When the workload has a high percentage of writes, such as a read-write ratio of 1, the algorithms that have the fewest number of replicas consume less energy because write requests take a high percentage of the overall energy consumption. The simulation results in Figure 4.6 demonstrate FDRM has the lowest energy consumption when the read write ratio is 1. As the read write ratio increases from 10 to 40, FDRM consumes more energy than other algorithms. There are two reasons. First, the addition and deletion of replicas in FDRM only considers the workload and does not consider the hop distance for each data access. This approach is not suitable for MANETs. Second, FDRM sends many broadcast messages to update the replica set information. Broadcasting messages in MANETs consumes large amounts energy. We also observe that ARAM consumes less energy than the other algorithms when the read-write ratio is between 10 and 40. There are two reasons. First, ARAM does not send agreement messages in terms of adding and or deleting replicas while agreement messages must be sent in FDRM and MFDRM. Second, ARAM minimizes hop distances of a replica update path, which reduces the energy consumption in terms of replica updates.

4.3.2 The effects of pause time

In this experiment, pause time is varied from 0 second to 400 seconds, which simulates scenarios from a high mobility to a low mobility. Table 4-3 lists the parameter settings in

this experiment. Figure 4-7 shows the difference between the initial number of replicas in the system and the final number of replica in the system after the simulation, which reflects whether the replica set expands or shrinks during the simulation. Figures 4-8 to 4-12 show the simulation results of all algorithms under different pause times.

Parameters	Value
Pause time	0, 100, 200 and 400 seconds
Transmission range	230 meters
Number of nodes	30
Number of replicas at the beginning of simulation	10
Number of traffic source nodes	20
Packet sending rate	1 packet / 2 seconds
Read write ratio	10 : 1

Table 4-3 experiment parameters in pause time

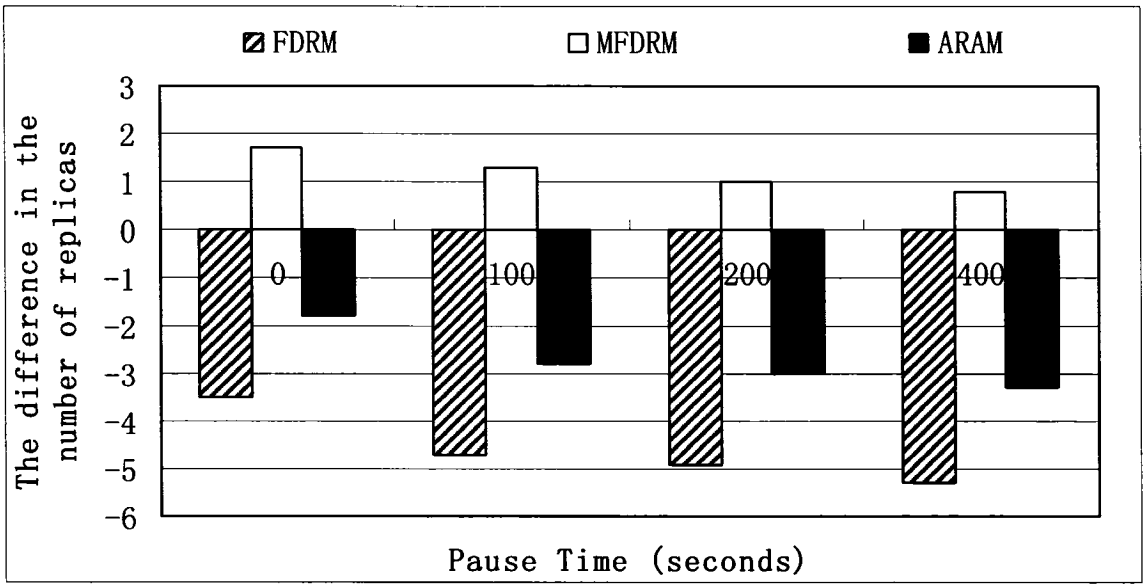

Figure 4.7 Effects of pause time on the difference in the number of replicas

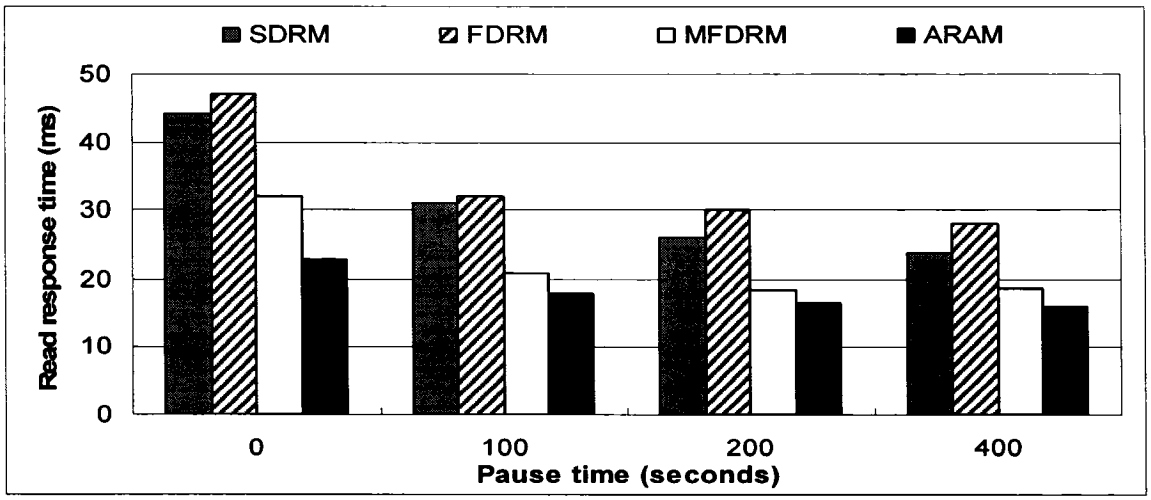

Figure 4.8 Effects of pause time on read response time

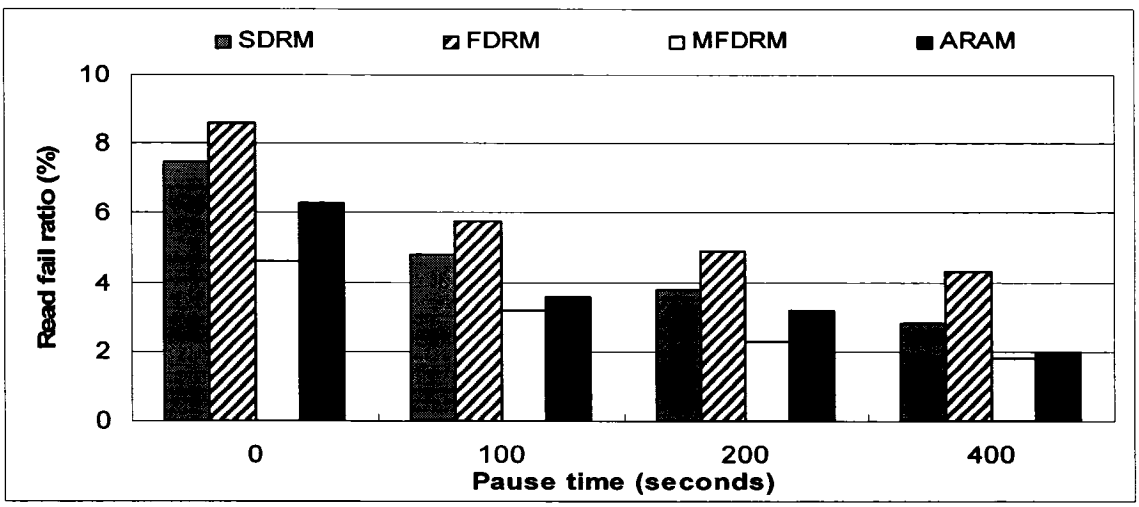

Figure 4.9 Effects of pause time on read fail ratio

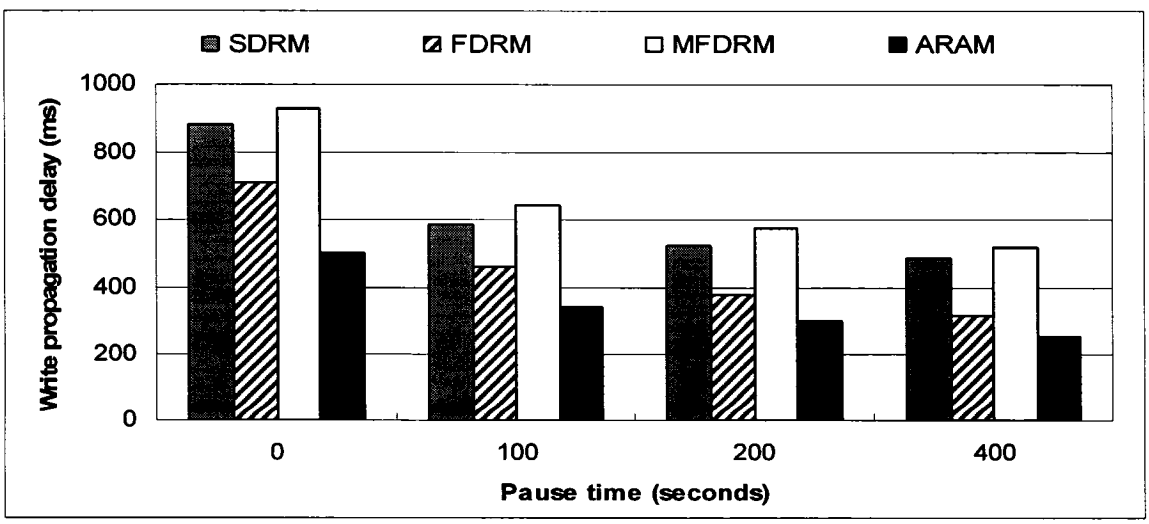

Figure 4.10 Effects of pause time on write propagation delay

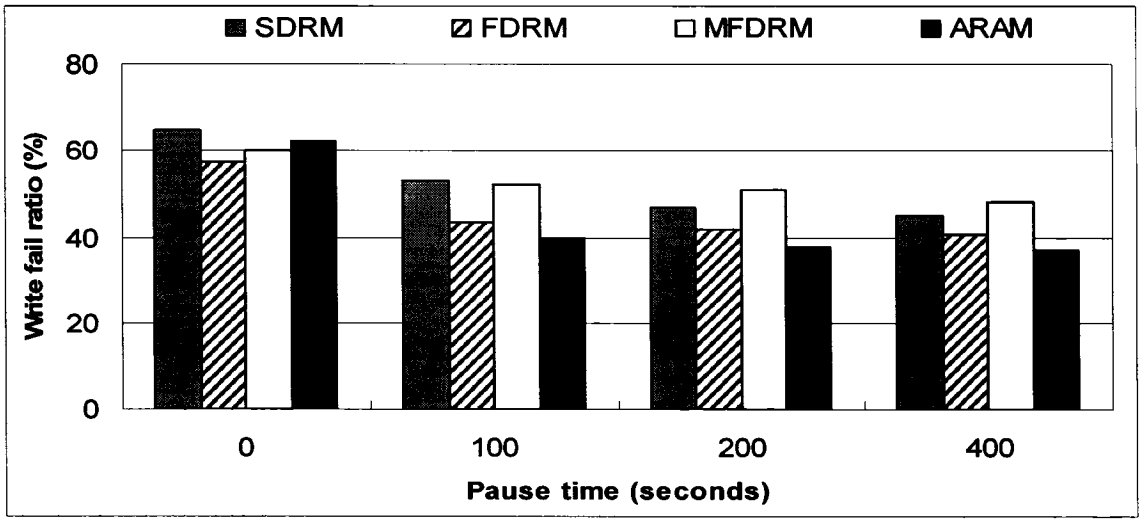

Figure 4.11 Effects of pause time on write fail ratio

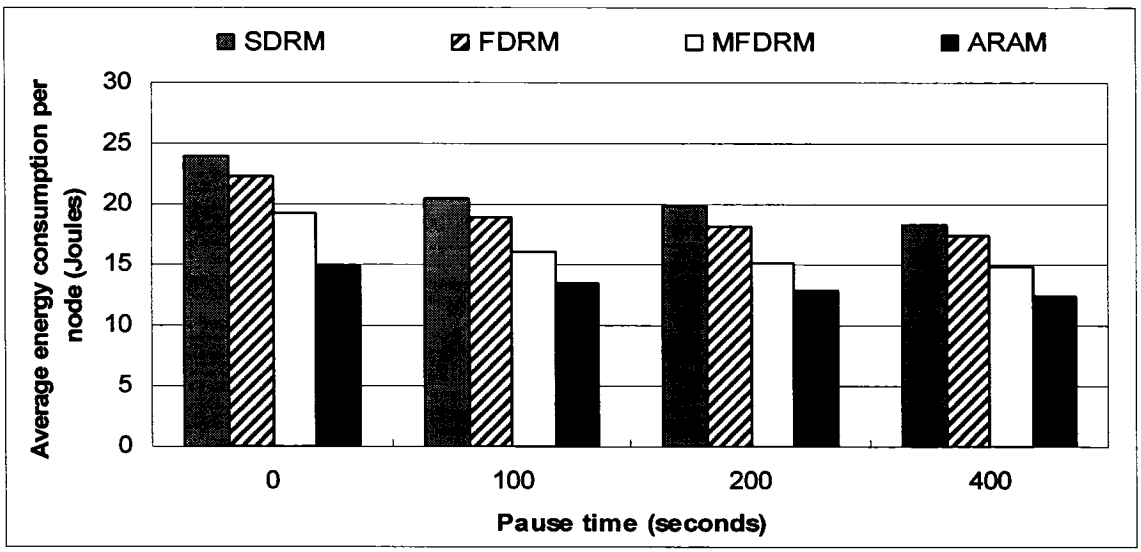

Figure 4.12 Effects of pause time on average energy consumption

From Figure 4.7, we see the replica sets in FDRM and ARAM shrink while the replica set in MFDRM expands. The reasons are mainly related to the read-write ratio, the traffic source nodes, the number of replica nodes and the hop distance for each data access. When the pause time is large enough, the system becomes stable. A replica node *k* may receive 20 writes because the number of traffic source nodes is 20. The reads that a replica node *k* receives from itself is 10 because the read-write ratio is 10. From (3.2), we see replica nodes in FDRM tend to be deleted during the simulation

because more writes are received than reads. Based on our simulation data and analysis of (3.7), we find $\sum_{i \in F} WT(i)d(i,s)$, the write communication cost between the replica node s and other replica node i ($i \notin s$), is the main reason that the replica set in ARAM shrinks. At the beginning of simulation, the number of replica nodes in the system equals 10. *WT(i)*, the number of writes that a replica node i receives during a time period t, may equal 20 because 20 traffic sources generate writes. The summation of $\sum_{i \in F} WT(i)d(i,s)$, including the hop distance for each write request, is obviously larger than the read and write cost that a single node *s* received. Therefore, replica nodes tend to be deleted and the number of replicas after simulation is less than 10. MFDRM does not shrink its replica set in this scenario. There are two reasons. First, MFDRM has a restricted replica deletion condition as we explained in Section 4.3.1. Second, some nodes during the simulation may receive more reads than writes because of the high write fail ratio. Therefore the replica set in MFDRM has a slight expansion. As the pause time becomes small, the reads/writes that can be delivered to the destination nodes decreases because nodes keep moving. A replica node *k*, therefore, receives fewer reads and writes. The decrease in writes received by a replica node *k* however, is larger than the decrease in reads received by a replica node *k* because a large portion of writes are generated from nodes that are far away from replica node *k* while reads are generated from local or non-replica nodes that are close to node *k*. The number of deleted nodes, therefore, decreases slightly or equivalently the number of added nodes increases slightly as the pause time becomes small.

In Figures 4.8 and 4.9, both read response time and read fail ratio decrease as the pause time increases. The major reason is that the network becomes more stable as the pause time increases. Finding a route for a read request, therefore, takes less time and

the chances of a network partition decreases. FDRM has the largest response time and highest read fail ratio because it has the fewest number of replicas. MFDRM has the lowest read fail ratio. This is because (a) the replica set of MFDRM expands during the simulation and it has the largest number of replicas among all algorithms. A scheme with more replicas during the simulation tends to reduce the read fail ratio; and (b) replicas in MFDRM can be expanded to any node not just the neighboring nodes. This approach distributes replicas more widely and therefore increases the replica availability, especially when a network partition takes place. The number of replicas in ARAM shrinks during the simulation. In Figure 4.8, ARAM has the lowest read response time among all algorithms. This is interesting because MFDRM has more replicas than ARAM. This leads us to suggest that when the network has high mobility, having more replicas in the system does not guarantee improving the performance of read operations.

High node mobility causes more frequent route failures and packet loss. The routing protocol needs to find new routes. Finding new routes increases control packet retransmission, which increases the propagation delay of replica updates. High node mobility also increases the probability of a network partition. The fail ratio of replica updates increases as networks are partitioned. Figure 4.10 shows ARAM has the lowest write propagation delay. There are two reasons for this. First, the replica set of ARAM shrinks during the simulation. A scheme that has fewer replicas tends to have a shorter write propagation delay. Second, ARAM minimizes the hop distance of a replica update path and replicas are expanded only to neighboring nodes, which makes replica nodes close to each other. As fewer hop distances are traveled for each replica update, the write propagation delay is reduced. This is more obvious when the network has a high mobility such as a pause time equals 0 second. Although the replica set of FDRM shrinks and FDRM has the lowest number of replicas, the simulation result shows its

write propagation delay is higher than that of ARAM. This also demonstrates that the number of replicas is not the only factor that affects the write propagation delay and locations of these replicas also affects write propagation delay.

Simulation results in Figure 4.11 show that both SDRM and ARAM have higher write fail ratios than FDRM and MFDRM when mobility is high. The reason is that SDRM is static and it is not suitable for high mobility networks. The update procedure in ARAM is also not suitable for high mobility networks in terms of its write fail ratio. A replica update request in ARAM is first sent to the closest replica node, which forwards the update request to other replicas nodes. In a high mobility environment, it is very likely that the first replica node may not receive the replica update request and, therefore, cannot forward the replica update request to other nodes. This update procedure in ARAM increases the replica update fail ratio when the network has high mobility. When the mobility is low, ARAM has the lowest write fail ratio due to the fact that its replica set shrinks and it minimizes the total hop distance of the update path. A scheme that has smaller hops distances in the replica update paths tends to reduce the replica update fail ratio as well.

The packet retransmission decreases as node mobility decreases. The average energy consumption for all mobile nodes in Figure 4.12, which represents the total packets sent during the simulation, therefore decreases as well. SDRM has the highest energy consumption among all algorithms due to the fact that the number of replicas and their locations does not change during the simulation. ARAM has the lowest energy consumption among all algorithms. There are three reasons. First, the replica allocation in ARAM considers not only workload but also the hop distance of each data access. This approach minimizes the overall communication cost in terms of replica reads and

replica writes. Second, ARAM minimizes the total hops in replica update paths. Third, ARAM does not send agreement messages in terms of replica addition and deletion. Energy consumption in FDRM is higher than energy consumption in MFDRM. There are two reasons. First, replica allocation in FDRM only considers workload while replica allocation in MFDRM considers both hop distance and workload. Second, more broadcast messages are sent in FDRM to update the replica set information in each node. Broadcasting in MANETs consumes large amounts of energy.

4.3.3 The effects of packet sending rate

In this experiment, packet sending rates are varied from 0.5 packet/s to 2 packet/s, which simulates scenarios from a low workload to a high workload. Table 4-4 lists the parameter settings in this experiment. Figure 4.13 shows the difference between the initial number of replicas in the system and the number of replicas in the system after the simulation. Figures 4.14 to 4.18 shows the simulation results of all algorithms under different packet sending rates.

Parameters	Value
Packet sending rate	0.5, 1.0, 1.5, 2.0 packet/second
Pause time	50 seconds
Transmission range	230 meters
Number of nodes	30
Number of replicas at the beginning of simulation	10
Number of traffic source nodes	20
Read-write ratio	10 : 1

Table 4-4 Experiment parameters in packet sending rate

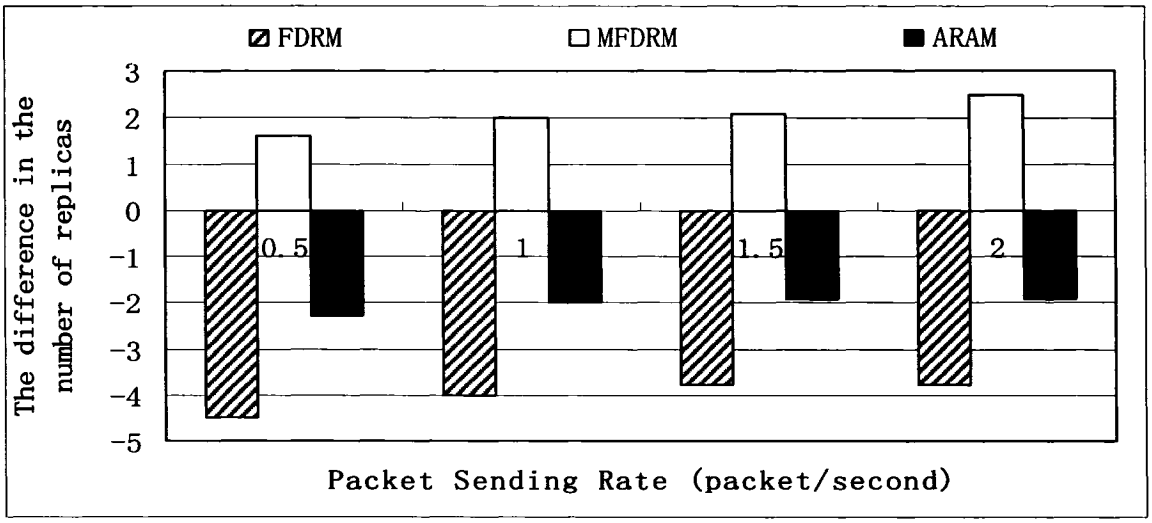

Figure 4.13 Effects of packet sending rate on the number of replicas

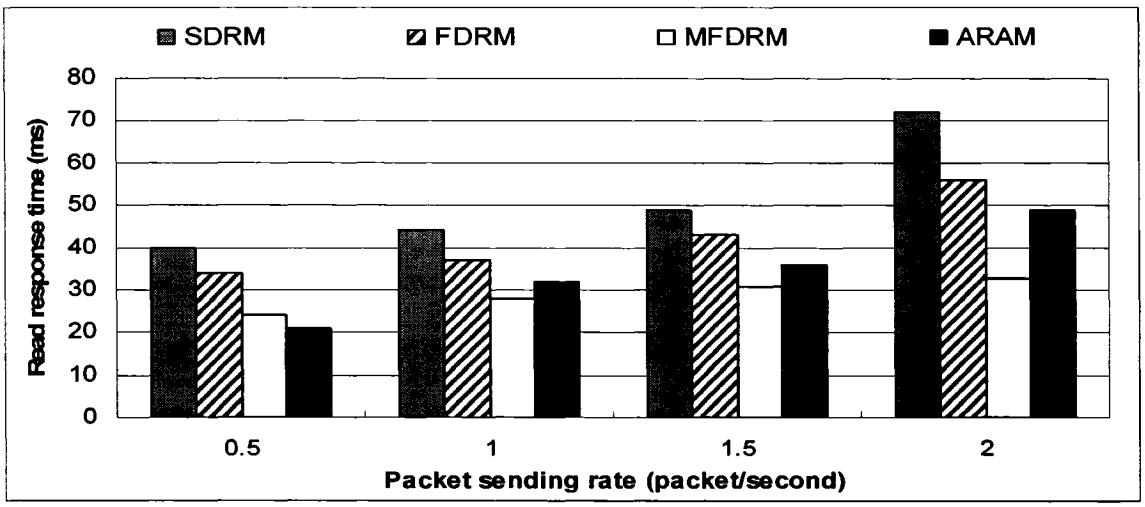

Figure 4.14 Effects of packet sending rate on read response time

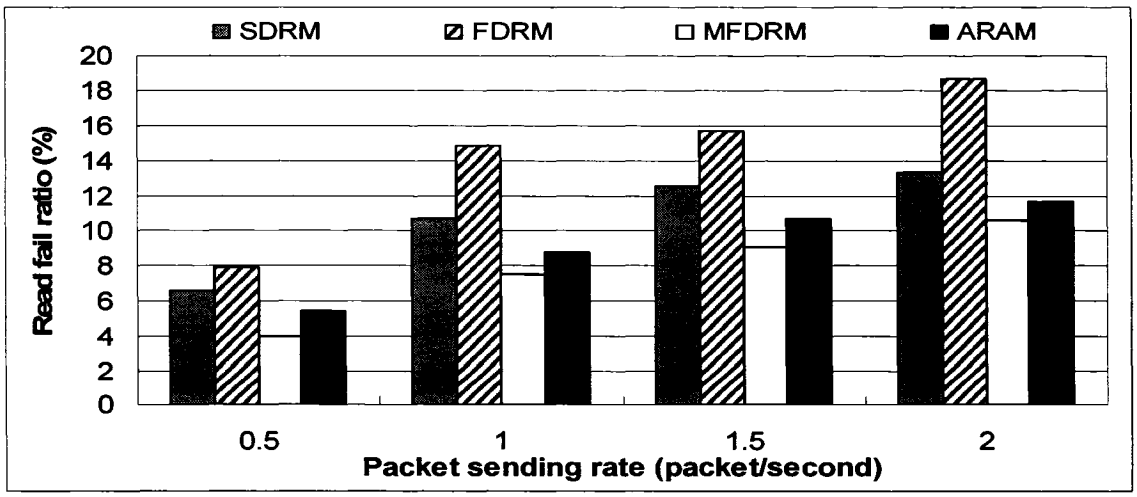

Figure 4.15 Effects of packet sending rate on read fail ratio

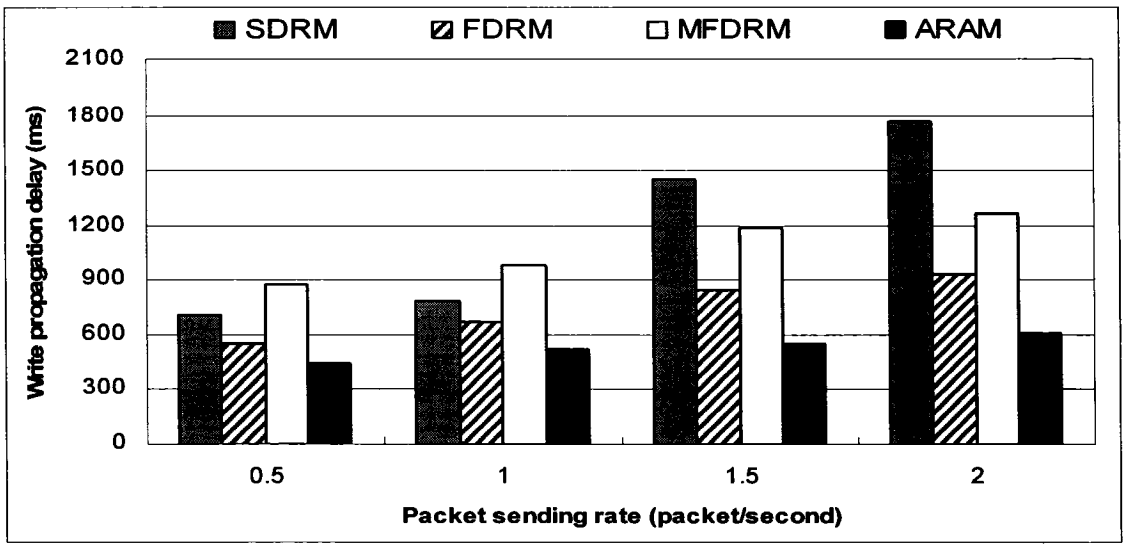

Figure 4.16 Effects of packet sending rate on write propagation delay

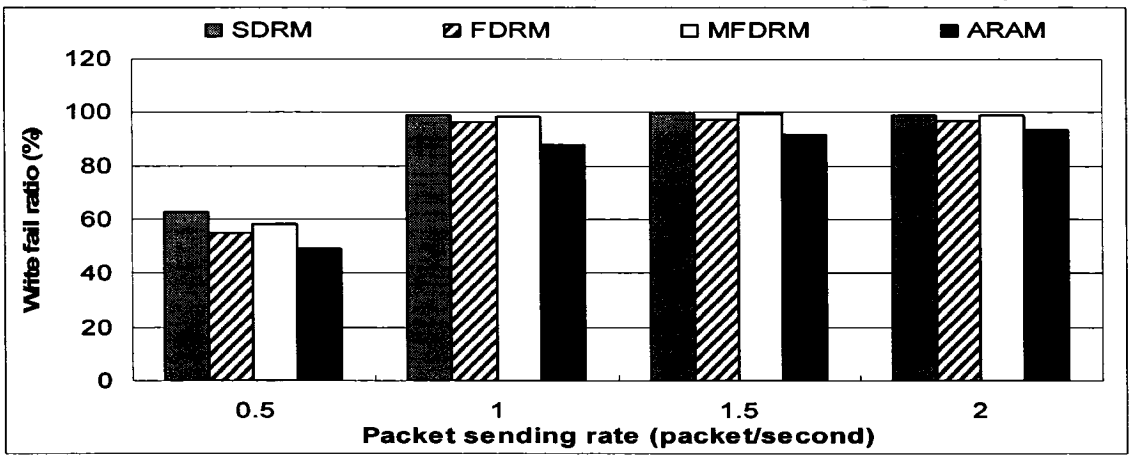

Figure 4.17 Effects of packet sending rate on write fail ratio

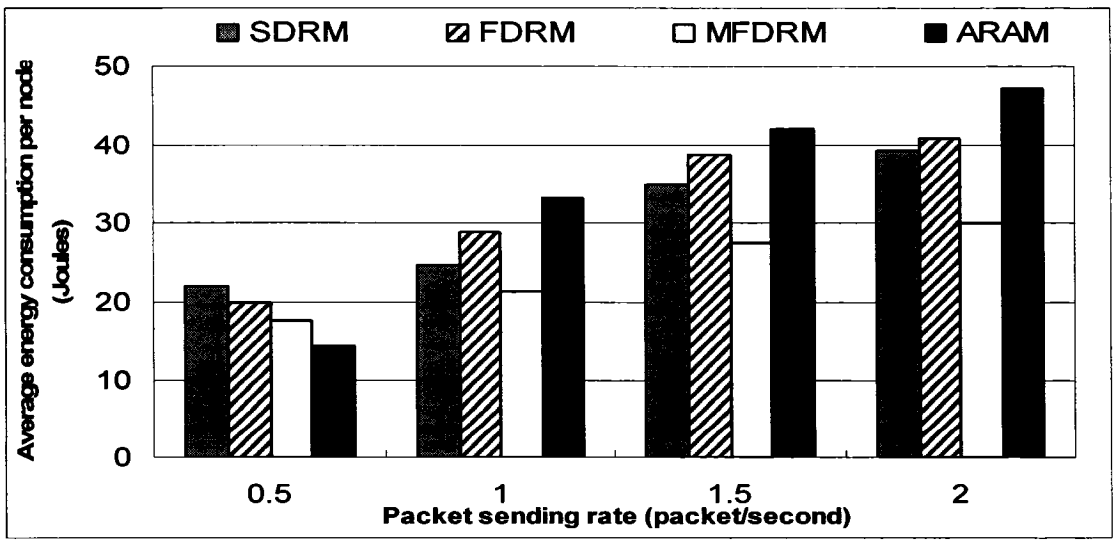

Figure 4.18 Effects of packet sending rate on average energy consumption

The simulation results in Figure 4.13 shows that the replica sets in FDRM and ARAM shrink while the replica set in MFDRM expands. The major reason is the same as in section 4.3.2. As the packet sending rate increases, a replica node *k* receives more reads and writes. The increase in writes received by replica node *k*, however, is smaller than the increase in reads received by replica node *k* because most writes take longer paths than reads to reach destination nodes. As the workload increases, a request that takes a longer path tends to get lost. The number of deleted nodes, therefore, decreases slightly as the packet sending rate increases.

Network traffic increases as the packet sending rate increases. High network traffic causes network resource contention and increases the end-to-end packet delay and packet loss. Figures 4.14 and 4.15 show that the read response time and the read fail ratio increase as the packet sending rate increases. MFDRM has the lowest read response time and read fail ratio because its replica set expands during the simulation. MFDRM has the largest number of replicas among all algorithms. The replica sets in FDRM and ARAM shrink during the simulation and ARAM has more replicas than FDRM. The read response time and read fail ratio of FDRM, therefore, are higher than the read response time and read fail ratio of ARAM.

Simulation results in Figures 4.16 and 4.17 shows that write propagation delay and write fail ratio increase as the packet sending rate increases. There are two reasons for this performance. First, the end-to-end delay and packet loss increases as more packets are sent during the simulation. Second, the number of replicas created increases slightly or the number of deleted replicas decreases slightly. The write fail ratio of all algorithms

is very high (85% - 99%) when the packet sending rate is 1 packet/second. The reasons are the same as in Section 4.3.1.

ARAM has the lowest write fail ratio and write propagation delay among all the algorithms. The replica set of ARAM shrinks during the simulation therefore it has fewer replicas to be update compared with SDRM and MFDRM. ARAM also minimizes the total hop distance in the replica update path. Simulation results also demonstrate that dynamic replication algorithms have lower write propagation delay compared with static replication when the network has high workloads. MFDRM has a higher write propagation delay than FDRM because its replica set expands during the simulation while the replica set of FDRM shrinks during the simulation.

Simulation results in Figure 4.18 show that average energy consumption for all nodes increases as the packet sending rate increases. ARAM has the largest energy increment when the packet sending rate is high. The reason is that the replica allocation interval is shortened when the packet sending rate is high. The replica allocation in ARAM is based on global data access information. Collecting data access information from other nodes consumes a large amount of energy because of broadcasting information. The replica set of FDRM shrinks during the simulation and its replica set changes more frequently, which results in more agreement messages and broadcast message to be set. FDRM, therefore, consumes more energy than MFDRM and SDRM.

4.3.4 The effects of network size

In this experiment, the number of nodes in the network is varied from 20 to 50. This simulates scenarios from a low node density to a high node density. Table 4-5 lists the parameter settings in this experiment. Figure 4.19 shows the difference between the

initial number of replicas in the system and the number of replicas in the system after the simulation. Figures 4.20 to 4.24 show the simulation results of all algorithms under different network sizes.

Parameters	Value
Number of nodes	20 , 30, 40 and 50
Pause time	50 seconds
Transmission range	230 meters
Number of replicas at the beginning of simulation	10
Number of traffic source nodes	20
Packet sending rate	1 packet / 2 seconds
Read write ratio	10 : 1

Table 4-5 experiment parameters in network size

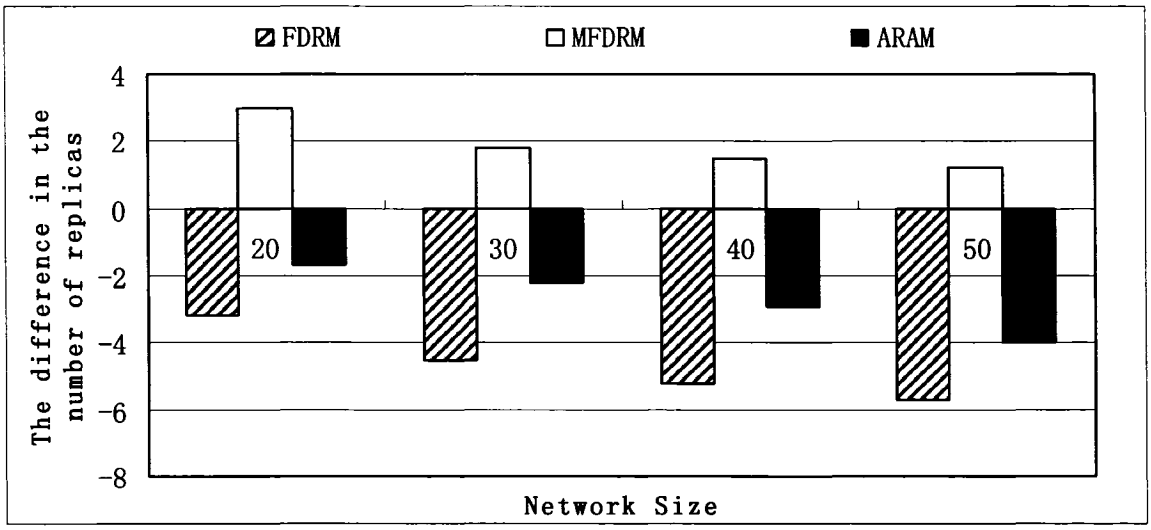

Figure 4.19 Effects of network size on the difference in the number of replicas

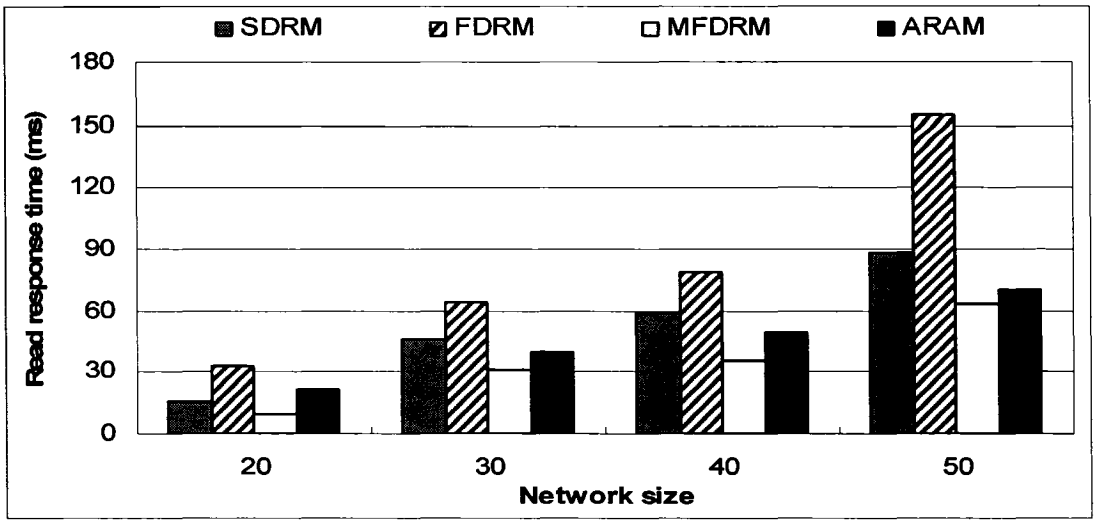

Figure 4.20 Effects of network size on read response time

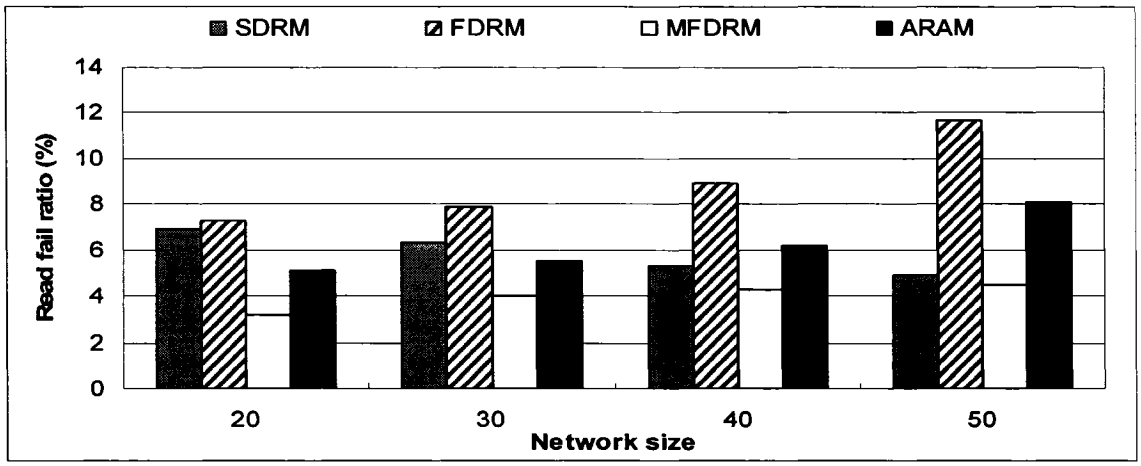

Figure 4.21 Effects of network size on read fail ratio

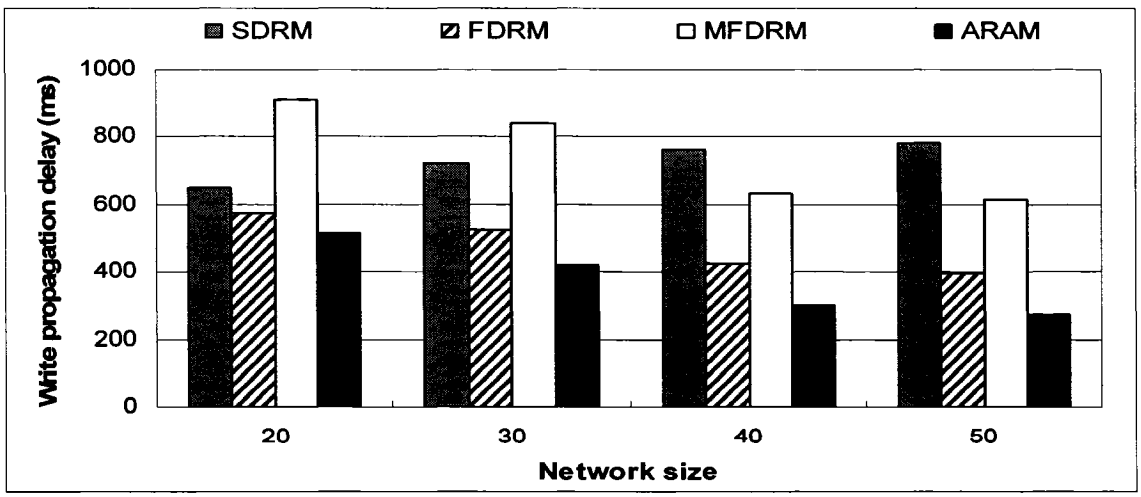

Figure 4.22 Effects of network size on write propagation delay

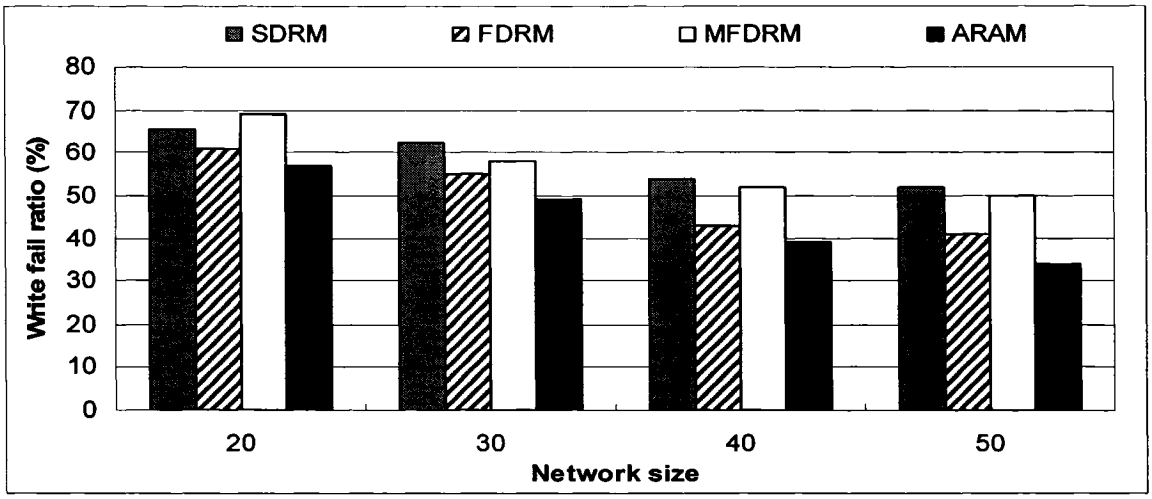

Figure 4.23 Effects of network size on write fail ratio

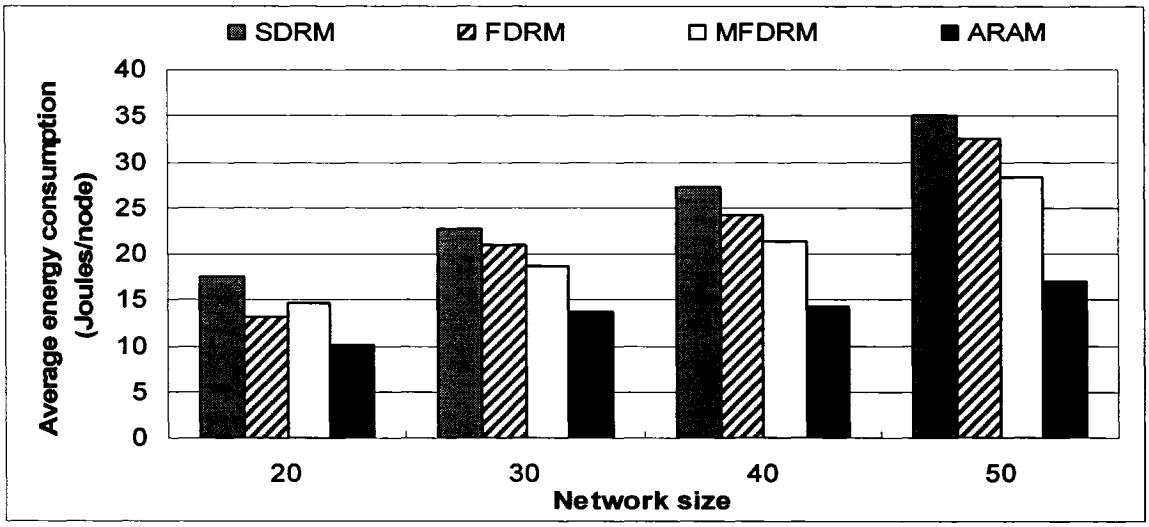

Figure 4.24 Effects of network size on average energy consumption

From Figure 4.19, we see the replica sets in FDRM and ARAM shrink while the replica set in MFDRM expands. The major reason is the same as in section 4.4.2. As the network sizes increase, the chance of network partition and packet loss decreases. A replica node *k*, therefore, can receive more reads and writes. The increases in writes received by a replica node *k* are larger than the increases in reads received by it. The reason is that a large portion of writes come from nodes that are far away from replica node *k* and the chances of these writes succeeding increases as the network density

increases. The number of deleted nodes increases or the number of added nodes decreases as the network size increases.

The network size is an indicator of node density inside the simulation area. High network size represents high node density. Simulation results in Figure 4.20 demonstrate that the read response time increases as the network size increases. There are two reasons for this performance. First, the probability of network partitions is higher in MANETs than in wired networks because of mobility. As the network density inside the simulation area increases, the probability of nodes being within transmission range increases. The probability of network partition therefore decreases. Both reads and writes that have long communication paths can be satisfied in this situation. These reads, which require many hops, increase the average read response time. Second, from Figure 4.19 we see the number of replicas created during the simulation decreases and the number of replicas deleted during the simulation increases as the network size increases. As few replicas are available during the simulation, the average read response time for replica reads increases.

Figure 4.21 shows that the read fail ratio of MFDRM, FDRM and ARAM increases while the read fail ratio of SDRM decreases. The number of replicas and their locations in SDRM does not change during the simulation. As the network density increases, the packet loss rate decreases. The read fail ratio in SDRM therefore decreases as the network size increases. For these dynamic replication algorithms, however, this is not the case because the number of deleted replicas increases as the network becomes dense. As more replicas are deleted during the simulation, the read fail ratio increases. FDRM has the highest read response time and read fail ratio while MFDRM has the

lowest read response time and read fail ratio. The reason is that MFDRM has the largest number of replicas while FDRM has the lowest number of replicas during the simulation.

The probability of a network partition decreases as the number of nodes increases in the simulation area. Write requests which take a long communication path, therefore, can reach the destination node. Simulation results in Figure 4.22 and 4.23 show that the write propagation delay of SDRM increases while the write fail ratio of SDRM decreases. The reason is that the number of replicas and their locations do not change during the simulation. As more write requests succeed, the write propagation delay increases and the write fail ratio decreases as well. However, the write propagation delay and write fail ratio decreases for the dynamic replication algorithms because the number of deleted replicas increases or the number of added replicas decreases. ARAM has the lowest write fail ratio and write propagate delay among all algorithms because its replica update procedure minimizes the total hop distance of write updates.

Figure 4.24 shows that the average energy consumption of all nodes increases as the network size increases. When the network becomes dense, the probabilities that reads and writes which take a long communication path can reach the destination node increases. This obviously increases the data packets and control packets to be sent during the simulation. ARAM has the lowest energy consumption because it minimizes the hop distance of the replica update path. Another reason is that ARAM does not send messages to control the number of replicas in the network. SDRM has the highest energy consumption because the number of replicas and their locations do not change during the simulation. FDRM has higher energy consumption than MFDRM when the network size is large. There are two reasons for this performance. First, replica allocation in FDRM only considers the data access load. This approach is not guaranteed to

minimize the overall communication cost. Second, more broadcast messages are sent in FDRM to coordinate and update the replica set information.

4.4 Summary

In this chapter, we compare the performance of the dynamic replication algorithms discussed in chapter 3 and a static replication algorithm SDRM. We conduct a series of simulations to investigate how each parameter of the simulation model affects the performance of different algorithms. Section 4.1 outlines our simulation model which consists of a mobility model, a workload model, an energy model and a network model. Section 4.2 illustrates five selected performance metrics and four selected factors that affect the performance of the algorithms. The five selected performance metrics are read response time, read fail ratio, write propagation delay, write fail ratio, and average energy consumption. The four selected factors that affect the performance of algorithms are read-write ratio, pause time, packet sending rate and network size. Section 4.3 discusses the simulation result and our analysis.

Simulation results demonstrate that FDRM responds well to a read-write ratio change. Its replica set expands or shrinks significantly when the workload has a high percentage of reads or writes, respectively. Simulation results also demonstrate FDRM has a higher read response time and read fail ratio than MFDRM when FDRM has more replicas than MFDRM. This demonstrates MFDRM outperforms FDRM in terms of read response time and read fail ratio. It also reflects that the actual increase in performance and availability through replication is a complex function of both the number of replicas and their placements in the system. Having more replicas in the system does not always reduce the read response time and the read fail ratio. We also see that MFDRM does not

shrink its replica set greatly when the workload has a large percentage of writes while ARAM does not expand its replica set greatly when the workload has a large percentage of reads. This demonstrates that the replica deletion condition in MFDRM and the replica addition condition in ARAM are too restrictive.

Simulation results also demonstrate that ARAM outperforms the other algorithms in write propagation delay and write fail ratio. The main reason is that the replica update procedure in ARAM minimizes the total hops of the update path. ARAM also has the lowest energy consumption in most simulation scenarios. There are two reasons. First, ARAM does not send agreement messages to control the number of replicas while FDRM and MFDR control the number of replicas. The drawback is that the number of replicas may become zero, which affects the availability. Second, replica allocation in ARAM is based on not only the number of reads and writes but also the hop distance for each read and write. This minimized the overall communication cost. However, when the access frequency is high, ARAM consumes more energy than other algorithms. The reason is that replica allocation in ARAM is based on information collected from other nodes and its local read-write information. High access frequency shortens the replica allocation intervals, which results in more data collection messages needing to be sent. It therefore brings significant overhead when the access frequency is high.

We realize that the write fail ratio is extremely high in MANETs. Maintaining replica consistency in MANETs is therefore extremely difficult when the network has a high workload. Simulation results also demonstrate that dynamic replication algorithms ARAM and MFDRM outperform the static replication in terms of energy consumption. FDRM which allocates replicas based on only the workload of reads and writes is not suitable for MANETs. Based on our simulation results, we rank the dynamic replication

algorithms FDRM, ARAM and MFDRM under different system features in Table 4-6. In the next chapter, we present our suggestions and future work.

System Features	High workload				
Algorithms	Performance Metrics				
	Write propagation delay	Write fail ratio	Energy Consumption	Read response time	Read fail ratio
ARAM	Good	Good	Poor	Fair	Fair
FDRM	Fair	Fair	Fair	Poor	Poor
MFDRM	Poor	Poor	Good	Good	Good
System Features	Low workload				
Algorithms	Performance Metrics				
	Write propagation delay	Write fail ratio	Energy Consumption	Read response time	Read fail ratio
ARAM	Good	Good	Good	Fair	Fair
FDRM	Fair	Fair	Poor	Poor	Poor
MFDRM	Poor	Poor	Fair	Good	Good
System Features	High mobility				
Algorithms	Performance Metrics				
	Write propagation delay	Write fail ratio	Energy Consumption	Read response time	Read fail ratio
ARAM	Good	Poor	Good	Good	Poor
FDRM	Fair	Good	Poor	Poor	Fair
MFDRM	Poor	Fair	Fair	Fair	Good
System Features	Low mobility				
Algorithms	Performance Metrics				
	Write propagation delay	Write fail ratio	Energy Consumption	Read response time	Read fail ratio
ARAM	Good	Good	Good	Good	Fair
FDRM	Fair	Fair	Poor	Poor	Poor
MFDRM	Poor	Poor	Fair	Fair	Good

System Features	Equal read write ratio				
Algorithm	Performance Metrics				
	Write propagation delay	Write fail ratio	Energy consumption	Read response time	Read fail ratio
ARAM	Fair	Good	Fair	Fair	Fair
FDRM	Good	Fair	Good	Poor	Poor
MFDRM	Poor	High	Poor	Good	Good
System Features	Low percentage of write requests				
Algorithms	Performance Metrics				
	Write propagation delay	Write fail ratio	Energy consumption	Read response time	Read fail ratio
ARAM	Good	Good	Good	Poor	Poor
FDRM	Poor	Poor	Poor	Fair	Fair
MFDRM	Fair	Fair	Fair	Good	Good

Table 4-6 Ranking of ARAM, FDRM and MFDRM under different system features

Chapter 5
Conclusions and Future work

The characteristics of MANETs pose significant challenges for providing data services. Frequent network partitioning in MANETs decreases data availability and makes it difficult to maintain data consistency. Previous research demonstrates that data replication is a feasible solution to alleviate the problems caused by poor network connectivity and network partitions. Data replication, however, faces significant challenges in MANETs because of their lack of infrastructure and their dynamic network topologies.

The dynamic network topologies of MANETs require dynamic replication algorithms. Unfortunately, only a few replication algorithms, each emphasizing different aspects, have been proposed for MANETs. To the best of our knowledge, until now there has been no systematic comparison among different replication algorithms in MANETs. In this thesis, we use simulation to evaluate the dynamic replication algorithms FDRM [7], ARAM [8], MFDRM and a static replication algorithm SDRM. We believe studying the behaviors of these algorithms under various scenarios yields insights into the design of dynamic replication algorithms in MANETs.

5.1 Conclusions

The actual benefits achieved by data replication are a complex function of the number of replicas, their placement and the replica consistency protocol. Simulation results demonstrate that ARAM achieves the lowest energy consumption, replica update propagation delay and replica update fail ratio in most cases among all algorithms. A

drawback in ARAM is that no agreement protocol is executed when the number of replicas changes. This may lead to a situation in which the number of replicas drops to zero and affects data availability. ARAM does not respond well when the workload has a large portion of reads.

MFDRM outperforms other algorithms in terms of read response time and replica read fail ratio. MFDRM, however, suffers high replica write propagation delay and does not respond well when the workload has a large portion of writes. Compared with ARAM and MFDRM, FDRM is much simpler and adapts to the read-write ratio change in a timely manner. Simulation results, however, show replica allocation in FDRM is not accurate in MANETs because it fails to take into account the hop distance for each data access.

Simulation results demonstrate that dynamic replication algorithms, like ARAM and MFDRM, outperform static replication algorithms in most scenarios. We also realize that increasing the number of replicas in the system does not guarantee the proportional increment of availability and performance in read operations. On the other hand, an increase in the number of replicas in the system may cause severe problems in maintaining the replica consistency because of the dynamic nature of MANETs.

Simulation results lead us to propose the following guidelines in designing replication systems:

- The Read-One-Write-All replica consistency protocol is not suitable for MANETs because the write fail ratio in MANETS is very high. Furthermore, the number of updatable replicas in MANETs must be small for applications that have tight consistency requirements. Tight correctness and consistency requirements imply

lower performance and availability.

- A dynamic replication algorithm has lower write propagation delay compared with a static replication algorithm when MANETs have a high workload.
- Compared with a dynamic replication algorithm, which only considers the number of data accesses when making a decision in replica allocation, a dynamic replication algorithm, which considers both the number of data accesses and their hop distance, improves performance.
- Dynamic replication algorithms, which control the number of replicas among a certain range, consume a significant amount of energy because broadcasting agreement messages in MANETs generates a large amount of traffic.

5.2 Future Work

There are many issues related to dynamic replication in MANETs and it is impossible to address all of them. Through our simulations, we found the following issues need further study:

- Accurate hop distance information

  Most dynamic replication algorithms in MANETs assume the hop distance information between any two mobile nodes is available. This assumption is not realistic because of the dynamic nature of the ad-hoc network. An efficient way to collect accurate hop information in MANETs needs to be investigated in order to make the dynamic replication systems practical.

- Efficient broadcasting mechanism

  Most dynamic algorithms in MANETs uses blind flooding [27] to send broadcast information. Blind flooding generates huge amounts of network traffic. Dynamic replication algorithms need an accurate replica allocation policy to improve performance and availability. An accurate replica allocation policy usually requires

that each node has a global view of the system. Obtaining a global view of the system may cause lots of overhead in the system. An efficient broadcast mechanism in MANETs can thus be combined with dynamic replication in MANETs to minimize the message traffic. On the other hand, an efficient dynamic replication algorithm on MANETs needs to minimize information broadcasts.

- Replica update and reconciliation mechanism

  An efficient and reliable replica update approach must be found to maintain replica consistency in MANETs. Replica reconciliation mechanisms must be studied to solve the inconsistency between a node's local view of the current replica set and the global view of the current replica set.

- Replica consistency model

  Most existing dynamic replication algorithms use Read-One-Write-All protocol to maintain the replica consistency. Simulation results, however, demonstrate the write fail ratio is fairly high. Other replica consistency protocols, such as the Quorum Consensus method, which only updates a portion of all replicas, may be worth further investigation.

REFERENCES

[1] Homepage of *IETF MANETs* Working Group *http://www.ietf.org/html.charters/manet-charter.html*

[2] B. Leiner, D. Nielson, and F. A. Tobagi, "Issue in Packet Radio Network Design", in Proceedings of *IEEE Global Telecommunications Conference,* vol. 75, pp. 6-20, January 1987

[3] I. Chlamtac, M. Conti, and J. Liu, "Mobile Ad Hoc Networking: Imperatives and Challenges", in Proceedings of *Ad Hoc Network Journal,* Vol.1, No.1, pp. 13-64, July 2003

[4] T. Hara, "Effective Replica Allocation in Ad-hoc Networks for Improving Data accessibility", in Proceedings of *IEEE Conference on Computer and Communications*, pp. 1568-1576, April 2001

[5] T. Hare, "Replica Allocation in Ad-hoc Networks with Periodic Data Update", in Proceedings of *International Conference on Mobile Data Management*, pp.79-86, January 2002

[6] K Wang, B. Li, "Efficient and Guaranteed Service Coverage in Partitionable Mobile Ad-hoc Networks", in Proceedings of *IEEE Conference on Computer and Communications,* pp.1089-1098, June 2002

[7] X. Zhou, X. Lu, M. Hou and J. Wu "A Dynamic Distributed Replica Management Mechanism Based on Accessing Frequency Detecting", in Proceedings of *ACM Special Interest Group on Operating Systems Review* Volume 38, pp. 26-34, July 2004

[8] J. Zheng, Y. J. Wang, X. C. Lu, K. Yang "A Dynamic Adaptive Replica Allocation Algorithm in MANETs", in Proceedings of *the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pp. 65-70, 2004.

[9] A.A. Helal, A.A. Heddaya, and B.B. Bhargava, "Replication Techniques in Distributed Systems" Kluwer Academic Publishers 1996, ISBN: 0792398009

[10] D. K. Gifford, "Weighted voting for replicated data" in Proceedings of *the 7th Symposium on Operating System Principles*, pp. 150-162, December 1979

[11] Y. Bartal, A. Fiat, and Y. Rabani. "Competitive algorithms for distributed data management", in Proceedings of *the 24th ACM Symposium on Theory of Computing*, pp. 39-50, 1992

[12] G. Cabri, A. Corradi and F. Zambonelli "Experience of Adaptive Replication in Distributed File Systems", in Proceedings of the 22nd EUROMICRO Conference, pp. 39-50, 1996

[13] S. Acharya and S. B. Zdonik, "An Efficient Scheme for Dynamic Data Replication", Technical Report CS-93-43, Brown University, September 1993

[14] D. Barbara and H. Garcia - Molina, "Replicated Data Management in Mobile Environments: Anything New Under the Sun", in Proceedings of *IFIP WG 10.3 Working Conference on Applications in Parallel and Distributed Computing*, pp. 18-22, April 1994

[15] Q. R. Wang and J.F. Pâris, "Managing Replicated Data Using Referees", in Proceedings of *Workshop Mobility and Replication, European Conference on Object-Oriented Programming*, August 1995.

[16] D. Barbara, T. Imielinski, "Sleeper and Workholics: Caching Strategies in Mobile Environment", in Proceedings of the 1994 *ACM International Conference on Management of Data*, pp.1-12, June 1994

[17] J. Cay, K.L Tan, B. C. Ooi, "On Incremental Cache Coherency Schemes in Mobile Computing Environments", in Proceedings of *the Thirteenth International Conference on Data Engineering*, pp. 114-123, April 1997

[18] Y. Huang, S. Pistla, and O. Wolfson, "Data Replication for Mobile Computer", in Proceedings of 1994 *ACM International Conference on Management of Data*, pp.13-24, June 1994

[19] E. Pitoura, B. Bhargava, "Maintaining Consistency of Data in Mobile Distributed Environments", in Proceedings of *15th IEEE International Conference on Distributed Computing Systems*, pp. 404-413, 1995

[20] P. Sistla, O. Wolfson and Y. Huang, "Minimization of Communication Cost through Caching in Mobile Environments", in Proceedings of *IEEE Transactions on Parallel and Distributed Systems,* Vol. 9, No.4, April 1998.

[21] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad-hoc Network Routing Protocols", in Proceedings of *the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 85-97, October 1998.

[22] L. D. Fife and L. Gruenwald, "Research Issues for Data Communication in MANET Database Systems", in Proceedings of *the ACM Special Interesting Group on Management of Data Record*, Vol. 32, No. 2, pp. 42-47, June 2003

[23] J. L. Huang and M. S. Chen, "Exploring Group Mobility for Replica Allocation in a MANET," in Proceedings of the *12th ACM International Conference on Information and Knowledge Management*, November 2003

[24] W. C. Peng and M. S. Chen, "Developing Data Allocation Schemes by Incremental Mining of User Moving Patterns" in Proceedings of a *Mobile Computing System IEEE Transactions on Knowledge and Data Engineering*, 15(1):70–85, February 2003

[25] H. K. Wu, M. H. Jin, J. T. Horong, and C. Y. Ke "Personal Paging Area Design Based on Mobile's Moving Behaviors", in Proceedings of *IEEE Conference on Computer and Communications*, pp. 21-30, April 2001

[26] X. Hong, M. Gerla, G. Pei, and C.-C. Chiang, "A Group Mobility Model for Ad-hoc Wireless Networks", in Proceedings of *the 2nd ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, August 1999.

[27] Y. Yi, M. Gerla and T. J. Kwon "Efficient Flooding in Ad-hoc networks: a Comparative Performance Study", in Proceedings of *2003 IEEE International Conference on Communication*, Vol.2, pp. 1059-1063, May 2003

[28] G. Karumanchi, S. Muralidharan, and R. Prakash, "Information Dissemination in Partitionable MANETs," in Proceedings of *IEEE Symposium on Reliable Distributed Systems*, pp. 4-13, October 1999

[29] NS-2 network simulator: http://www.isi.edu/nsnam/ns

[30] D. Pong and T. Moors "The Impact of Random Waypoint Mobility on Infrastructure Wireless Networks", in Proceedings of *the 11th International Conference on Parallel and Distributed Systems Workshops*, pp. 140-144, 2005

[31] E. M. Royer and C-K Toh, "A Review of Current Routing Protocols for Mobile ad-hoc Wireless Networks", in Proceedings of *1999 IEEE Personal Communications,* pp. 46-55, April 1999

[32] C-K. Toh, "Mobile ad-hoc Wireless Networks: Protocols and Systems", I*SBN: 0130078174, Prentice Hall Publishers*, 2001.

[33] E. Pacitti and E. Simon, "Update propagation strategies to improve freshness in lazy master replicated databases", in Proceedings of *Very Large Data Bases Journal*, 8(3-4): pp. 305-318, 2000

[34] B. Kemme and G. Alonso, "A suite of database replication protocol based on group communication primitives", in Proceedings of *the 18th International Conference on Distributed Computing Systems*, pp. 1576-1586, May 1998

[35] M. Papadopuli and H. Schulzrinne, "Design and implementation of a peer-to-peer data dissemination and prefetching tool for mobile users", in Proceedings of *the First NY Metro Area Networking Workshop,* pp. 10 -21, March 2001

[36] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere, "Coda: A highly available file system for a distributed workstation environment", in Proceedings of *IEEE Transactions on Computers*, pp. 447-459, April 1990

[37] D. Terry, M. Theimer, K. Peterson, A. Demers, M. Spreitzer, and C. Hauser, "Managing update conflict in bayou, a weakly connected replicated storage system", in Proceedings of *the 15th ACM Symposium on Operating Systems Principles,* pp. 172-183, December 1995

[38] R. Guy, J. Heidemann, W. Mak, T. Page, G. Popek, and D. Rothmeier, "Implementation of the ficus replciated files system", in Proceedings of *USENIX Conference,* pp. 63-71, June 1990

[39] R. Guy, P. Reicher, D. Ratner, M. Gunter, W. Ma, and G. Popek, "Rumor: Mobile data access through optimistic peer-to-peer replication", in Proceedings of *the 17th international Conference on Conceptual Modeling*, pp. 254 - 265, November 1998

[40] T.D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems", in Proceedings of *Journal of the ACM*, 43(2), pp. 225 - 267, 1996

[41] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process", in Proceedings of *Journal the ACM,* 32(2), pp. 374 - 382, April 1985

[42] D. Ratner, "Roam: A Scalable Replication System for Mobile and Distributed Computing", PhD. Dissertation, University of California, 1998

[43] H. Yu, P. Martin and H. S. Hassanein, "Cluster - Based Replication for Large - Scale Mobile Ad - hoc Networks", in Proceedings of *the International Conference on Wireless Communications and Mobile Computing*, pp. 552-557 June 2005.

[44] D. B. Johnson, "Routing in Ad-hoc Networks of Mobile Hosts," in Proceedings of *the IEEE Workshop on Mobile Computing Systems and Applications*, pp. 158-163, December 1994.

[45] S. Corson and J. Macker, "MANETing (MANET): Routing Protocol Performance Issues and Evaluation Considerations", *Internet Engineering Task Force RFC 2501.*

[46] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," in Proceedings of t*he Special Interest Group on Data Communications Conference on Communications Architectures, Protocols and Applications*, pp. 234-244, August 1994.

[47] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in Proceedings of *the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90-100, February 1999.

[48] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad-hoc Network Routing Protocols," in Proceedings of *the Fourth Annual ACM/IEEE International Conference on Mobile and Networking*, pp. 85-97, October 1998.

[49] P. Jackquet, P. Muhlethaler, T. Cluusen, A. Laouiti, A. Qayyum, and L. Viennot,

"Optimized Link State Routing Protocol for Ad-hoc Network," *MANET (MANET) Working Group, Internet Engineering Task Force*, February 2000

[50] D. B. Johnson, D. A. Maltz, and J. Broch, "DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad-hoc Networks," *Ad-hoc Networking*, edited by Charles E. Perkins, Chapter 5, pp. 139-172. Addison-Wesley, 2000

[51] S. Lee, W. Su and M. Gerla, "On-Demand Multicast Routing Protocol (ODMRP) for Ad-hoc Networks", Internet Engineering Task Force MANET Working Group, Internet-draft, 2000.

[52] V. Park, and S. Corson, "Temporally-Ordered Routing Algorithm (TORA) Version 1 Functional Specification," Internet draft, Internet Engineering Task Force, 1997.

[53] J.J. Garcia-Luna-Aceves and M. Spohn, "Source-Tree Routing in Wireless Networks", in Proceedings of *International Conference on Network Protocols*, October 1999.

[54] R. Ogier, F. Templin and M. Lewis, "Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)", RFC 3684, Network Working Group, 2004.

[55] B. Bhargava, A. Helal, and K. Friesen, "Analyzing availability of replicated database systems", in Proceedings of *the International Journal of Computer Simulation*, 1(4), pp. 393-418, December 1991

[56] A. Kumar and A. Segev, "Optimizing voting-type algorithms for replicated data", in Proceedings of *the International Conference on Extending Database Technology: Advances in Database Technology*, pp. 428-442, March 1998

[57] E. J. Shekita and M. J. Carey, "Performance enhancement through replication in an Object-Oriented DBMS", in Proceedings of *ACM Special Interest Group on Management of Data Conference*, pp 325-336, June 1989

[58] I. S. Pressman, S. A. Cook and J. K. Pachl, "The Optimal Location of Replicas in a Network Using a read-one-write-all Policy", in Proceedings of *Journal of Distributed Computing*, Volume 15, pp. 57-66, April 2002

[59] J. Barreto, "Information Sharing in Mobile Networks: a Survey on Replication Strategies", *Technical Report, Inesc-ID*, September 2003.

[60] P. Reiher, G. Popek, M. Gunter, J. Salomone, and D. Ratner, "Peer-to-Peer Reconciliation Based Replication for Mobile Computers", in Proceedings of *European Conference on Object Oriented Programming 96 Second Workshop on Mobility and Replication*, June 1996.

[61] E. Pacitti, P. Minet and E. Simon, "Replica Consistency in Lazy Replicated Databases", in Proceedings of *Distributed and Parallel Databases*, Volume 9, Number 3, pp. 237-267, May 2001

[62] B. R. Badrinath and T. Imielinski, "Replication and mobility", in Proceedings of *the 2nd IEEE Workshop on Management of Replicated Data,* pp. 9-12, November 1992

[63] C.-K. Toh, "Ad Hoc Mobile Wireless Networks: Protocols and Systems" Prentice Hall, ISBN: 0130078174, Chapter 1&2, December 2001

[64] O. Wolfson, S. Jajodia, Y. Huang„ "An Adaptive Data Replication Algorithm", in Proceedings of *ACM Transactions on Database Systems*, Vol. 22, No. 2, pp. 255-314, June 1997

APPENDIX A – FDRM ALGORITHM

This section lists the details and pseudo code of the FDRM algorithm. Figure A.1 lists the variables and constants in FDRM. Figure A.2 lists different types of message in FDRM.

Variable/Constant	Definition
k	the replica node that executes the algorithm FDRM
R_{in}	the number of reads from node k itself during a time period T_n
$R_{out,j}$	the number of reads from node j *to node* k ($j \neq k$) during a time period T_n
W_{out}	the number of updates from other nodes during a time period T_n
W_{in}	the number of writes from node k itself during a time period T_n
F	the set of replica nodes in the system
outRead	a list whose entry represents the number of reads that node k receives. For example, *outRead(j)* means the number of reads, which are sent from node j, receives by node k
T_n	the time duration of the n^{th} replica allocation interval
A_n	the total number of data access during the time period T_n
N	the number of *RPCA_ADD_PERMIT* messages that replica node k receives
D	the number of *RPCA_DELETE_PERMIT* messages that replica node k receives
V	the set of all mobile nodes in the system
state	a variable indicating the state of node k
MAX_INTERVAL	the maximum replica allocation interval
MIN_INTERVAL	the minimum replica allocation interval
MAX	the maximum number of replicas
MIN	the minimum number of replicas
V	the set of all mobile nodes in the system

Figure A.1 Variables and constants in FDRM

Message Type	Message Functionality
RPCA_READ	query a replica
RPCA_READ_REPLY	a query response
RPCA_WRITE	a replica update message
_ADDING	coordinate the adding of a new replica to the system
RPCA_ADD_PERMIT	the permission of adding a replica from a single node
RPCA_ADDED	add a replica to the system
RPCA_ADDED_REPLY	a broadcast message informs a replica is added to the system
RPCA_DELETING	coordinate the deleting of a replica from the system
RPCA_DELETE_PERMIT	the permission of deleting a replica from a single node
RPCA_DELETED	a broadcast message indicating the replica is deleted
RPCA_SWITCH	switch a replica in the system
RPCA_SWITCH_REPLY	a broadcast message indicating a replica is switched in the system

Figure A.2 Messages in FDRM

AllocateRpca

AllocateRpca is executed on a replica node *k* when the replica allocation phase starts. Figure A.3 shows the pseudo code of *AllocateRpca*. Node *k* first executes the *AddRpca* and *DeleteRpca* procedures, as in lines 1 and 2. Next, if both add and/or delete a replica fail and the current number of replica reaches *MAX*, node *k* applies the migration process. Finally, node *k* executes procedure *NextInterval* to set the next replica allocation interval.

```
Begin AllocateRpca
1    execute AddRpca procedure
2    execute DeleteRpca procedure
3    if AddRpca and DeleteRpca fails and |F| = MAX then
4            for  each i ∈ outRead  do
5                 if  outRead[i] > R_in  then
6                      execute SwitchRpca procedure
9                      break;
10                end if
11           end for
12   end if
13   execute NextInterval procedure
End AllocateRpca
```

Figure A.3 *AllocateRpca* procedure in FDRM

AddRpca

If the current number of replicas in the system is less than the maximum number of replicas in the system, the add procedure starts. Replica node *k* searches the *outRead* list to find the first node *u* that satisfies (3.1), as in line 5. Next, node *k* changes its state from *Start* to *Adding,* as in line 6. Next, replica node *k* sends coordination messages *RPCA_ADDING* to other replica nodes to request permission to add a replica. A replica node sends a permission message *RPCA_ADDING_PERMIT* back to node *k* only when its state is *Adding*. We denote the number of permission messages that replica node *k* receives as *N* and the current number of replica is */F/*. If $N + |F| < MAX$, which means replica node *k* has enough permissions to add a new replica, replica node k changes its state from *Adding* to *Added*. Otherwise, it changes its state from *Adding* to *Start*. When a replica node *k* is in the *Added* state, it adds a replica to node *u* by sending message *RPCA_ADD*. When node *u* receives the new replica, it sends a broadcast message *RPCA_ADD_REPLY* to all nodes in the system to indicate that a new replica is added. Finally, node *k* changes its state from *Added* to *Start* when it receives message *RPCA_ADD_REPLY*.

```
Begin AddRpca
1   If  |F| < MAX  then
2       i → 0
3       for  each entry in outRead  do
4           u → outRead[i]
5           If  u satisfies 3.1 then
6                   state → Adding
7                   for  each node n, n ∈ F and n ≠ k  do
8                       node k sends message RPCA_ADDING to node n
9                   end for
10                  break;
11          end if
12          i → i + 1
13      end for
14      N → 0
15      for each received RPCA_ADDING_PERMIT do
16              N → N + 1
17      end for
18      If N + |F| < Max  then
19              state → Added
20              node k sends message RPCA_ADDED to node u
21      else
22              state → Start
23      end if
24      on received RPCA_ADDED_REPLY do
25              F = F ∪ {u}
26              state → Start
28      end do
29    end if
End AddRpca

Begin OnRecv_RpcaAdd
1       F = F ∪ {u}
2       node u sends broadcast message RPCA_ADDED_REPLY
End OnRecvRpcaAdd
```

Figure A.4 *AddRpca* procedures in FDRM

DeleteRpca

If node *k* satisfies 3.2, it changes its states from *Start* to *Deleting*. Next, it sends the coordination message *RPCA_DELETING* to other replica nodes to request permission to delete its replica, as in line 4. A replica node sends a permission message back to node *k* only when its state is *Deleting*. We denote the number of permission messages that replica node *k* in *Deleting* state receives as *D* and the current number of replicas as |F|. If */F/ - MIN > D*, which means replica node *k* has enough permissions to delete its replica,

replica node *k* changes its state from *Deleting* to *Deleted,* as in line 12. When a replica node *k* is in the *Deleted* state, it deletes its replica and sends the message *RPCA_DELETED* to inform all nodes in the system that its replica is deleted. Finally, it changes its state from *Deleted* to *Start*.

```
Begin DeleteRpca
1   If node k satisfies 3.2 then
2           state → Deleting
3           for each node n, n∈F and n≠k do
4               node k sends message RPCA_DELETING to node n
5           end for
6           D → 0
7           for each received message RPCA_DELETE_PERMIT do
8               D → D + 1
9           end for
10          if D < |F| - MIN then
11              F = F ∪ - {k}
12              state → Deleted
13              node k broadcasts message RPCA_DELETED
14          end if
15          state → Start
16  end if
End DeleteRpca

Begin OnRecv_RpcaDeleted
1   F = F ∪ - {k}
End OnRecv_RpcaDeleted
```

Figure A.5 *DeleteRpca* procedures in FDRM

SwitchRpca

Node *k* searches the *outRead* list to find the first node *u* that satisfies (3.4) and changes its state from *Start* to *Switch*. When replica node *k* is in the *Switch* state, it sends a message *RPCA_SWITCH* to node *j*. Upon receiving *RPCA_SWITCH* from replica node *k*, node *j* adds a replica to its local cache and sends a broadcast message *RPCA_SWITCH_REPLY* to all nodes in the system to indicate the switching of replica from node *k* to node *j*. Finally, when node *j* receives *RPCA_SWITCH_REPLY*, it changes its state from *Switch* to *Start*.

```
Begin SwitchRpca
1    for  each outRead[i]  do
2         u  →  outRead[i]
3         if  R_out, u > R_in  then
4              state  →  Switch
5              node k sends message RPCA_SWITCH to node u
5              break;
6         end if
7         i  →  i + 1
8    end for
9     On receive RPCA_SWITCH_REPLY do
10        F = F ∪ - {k} + {u}
11        state  →  Start
End SwitchRpca

Begin OnRecv_RpcaSwitch
1    node u sends broadcast message RPCA_SWITCH_REPLY
2    F = F ∪ - {k} + {u}
End OnRecv_RpcaSwitch
```

Figure A.6 *SwitchRpca* procedures in *FDRM*

NextInterval

The next replica allocation interval is calculated based on 3.3, as in line 6. Some special cases are addressed as well. If there are no data accesses during the current interval, then no matter how many accesses the replica had in the last interval, the length of the next interval is twice the length of the current interval, as in line 2. On the other hand, if there are no data accesses during the last interval, then no matter how many data access in current interval, the length of next interval is half of the length of current interval, as in line 4. To avoid the length becoming too short or too long, a lower and upper limit is set, as in line 9 and line 11.

```
Begin NextInterval
1    if  A (n) == 0   then
2            T(n+1) = 2 T(n)
3    else if  A (n-1) == 0   then
4            T(n+1) = 0.5 T(n)
5    else
6            T (n+1) →T (n) * A (n-1) * T (n) / (T (n-1)*A (n))
7    end if
8    if  T (n+1) > MAX_PERIOD      then
9            T (n+1) = MAX_PERIOD
10   else if  T (n+1) < MIN_PERIOD    then
11           T (n+1) = MIN_PERIOD
12   end if
13   return  T (n+1)
End NextInterval
```

Figure A.7 *NextInterval* procedure in *FDRM*

ReadRpca

If node *k* has a replica in its local cache, node *k* increase the counter of local read r_{in} by 1 and the read operation succeeds locally. Otherwise, node *k* sends a message *RPCA_READ* to a replica node *n* which has the fewest hops to node *k*. When it receives the *RPCA_READ* message from node *k*, if node *n* never receives a request from node *k*, it inserts a new entry *outRead(k)* into the *outRead* list and set its counter to 1. If node n receives the request from node *k* before that, it increases its counter by 1. Then, node *n* moves *outRead(k)* to the head of the *outRead* list. The read operation succeeds when node *k* receives the message *RPCA_READ_REPLY* from node *n*.

```
Begin ReadRpca
1   if node k ∈ F then
2       r_in → r_in + 1
3       return O
4   else
5       n → min_{j∈F} d(k, j)
6       node k sends message RPCA_READ to node n
7       if node k receives message RPCA_READ_REPLY then
8           return O
9       end if
10  end if
End ReadRpca

Begin OnRecv_ReadRpca
1   if k ∉ in outRead then
2       outRead(k) → 1
3   else
4       outRead(k) → outRead(k) + 1
5   end if
6   node n moves k to the first entry in outRead
6   node n sends message RPCA_READ_REPLY to node k
End OnRecv_ReadRpca
```

Figure A.8 *ReadRpca* procedures in *FDRM*

UpdateRpca

If node *k* has the replica in it local cache, node *k* increase the counter of local write W_{in} by 1. Next, node *k* send message *RPCA_WRITE* to other replica nodes. When it receives the *RPCA_WRITE* message from node *k*, a replica node *n* increases its counter w_{out} by 1 and update its replica.

```
Begin UpdateRpca
1   if node k ∈ F then
2       w_in → w_in + 1
3       update replica O
4   end if
5   for each node n ∈ F and n ≠ k do
6       node k sends message RPCA_WRITE to node n
7   end for
End UpdateRpca

Begin OnRecv_UpdateRpca
1   w_out → w_out + 1
2   update replica O
End OnRecv_UpdateRpca
```

Figure A.8 *UpdateRpca and OnRecv_UpdateRpca* procedures in FDRM

APPENDIX B – ARAM ALGORITHM

This section lists the details and pseudo code of the ARAM algorithm. Figure B.1 lists the variables and constants in ARAM. Figure B.2 lists different types of messages in ARAM.

Variable/Constant	Definition
k	the replica node that executes the algorithm ARAM
V	all mobile nodes in the system
F	replica nodes in the system
$d(k, j)$	the shortest hop distance between node k and node j
$q(k)$	$q(k) = \sum_{j \in F} d(k, j)$ is defined as the weight value of a replica node k
$R(k, u)$	reads received by node k from node u during a time period t
$W(k, u)$	writes received by node k from node u during a time period t
$C(j)$	$C(j) = \{i \mid i \in F, d(j, F) = \min_{k \in F} d(j, k)\}$
A	$A = \{i \mid i \in V-F, k \in C(i)$, u is one of the shortest paths between i and $s\}$.
$R(A)$	reads from set A
$W(A)$	writes from set A
$WT(k)$	writes that replica node k receives from other nodes and itself
$WA(K)$	writes that replica node k received from set A
W	writes from set V
$\Delta change$	$\Delta change = \sum_{i \in A} W(i)(q(k) - q(Pi))$
B	$B = \{i \mid i \in V - F, k = p_i$ and $C(i) - \{k\} \neq \phi\}$
p_i'	for $i \in B$, $p_i' \in C(i)$ and $q(p_i') = min\ q(j), j \in C(i)-\{k\}$.
Z	$z \in F - \{s\}$ and $d(s, F - \{s\}) = d(s, z)$
w	$w \in F - \{k\}, q(w) - d(w, k) = max(q(i) - d(i, k)), i \in F - \{k\}$.

B.1 Variables and constants in FDRM

Message Type	Message Functionality
RPCA_READ	query a replica
RPCA_READ_REPLY	response from replica query
RPCA_UPDATE	update a replica
RPCA_FORWARD	forward a replica update request to the node that has the least weight
ALLOCATE	a message broadcast to collect data access information
ALLOCATE_REPLY	a reply message indicating the data access information at source node
RPCA_ADD	add a replica to the system
RPCA_ADDED	a broadcast message indicates a replica is added
RPCA_SWITCH	switch a replica
RPCA_SWITCHED	a broadcast message indicating the replica switch information
RPCA_DELETED	a broadcast message indicating the replica has been deleted

Figure B.2 Messages in ARAM

AllocateRpca

Figure B.3 has the detailed pseudo code of procedure *AllocateRpca*. First, node *k* sends a broadcast message *ALLOCATE* to collect data access information from other nodes. Upon received *ALLOCATE*, a node sends *ALLOCATE_REPLY* which contains the data access information during the current interval. Second, for each node *u*, which is a neighbor of node *k*, if node *u* satisfies 3.6, then procedure *AddRpca* is executed. If node *u* satisfies 3.8, then procedure *SwitchRpca* is executed. If node *u* satisfies 3.7, then procedure *DeleteRpca* is executed. Finally, procedure *NextInterval* is executed to set the next replica allocation interval.

```
Begin AllocateRpca
1   node k sends broadcast message ALLOCATE
2   on receive message ALLOCATE_REPLY do
3      for each neighbor node u of k and u ∉ F do
4            if u satisfies 3.6 then
5                 execute AddRpca procedure
6            end if
7            if u satisfies 3.8 then
8              execute SwitchRpca procedure
9            end if
10           if u satisfies 3.7 then
11             execute DeleteRpca procedure
12           end if
13     end for
14  execute NextInterval procedure
End AllocateRpca

Begin OnRecv_Allocate
1   node m sends a message ALLOCATE_REPLY
End OnRecv Allocate
```

Figure B.3 *AllocateRpca* procedures in ARAM

AddRpca

Node k sends a message *RPCA_ADD*, which contains the replica. When node *u* receives *RPCA_ADD* message, it sends a broadcast message *RPCA_ADDED* to inform all nodes in the system that a new replica is added.

```
Begin AddRpca
  1    node k sends message RPCA_ADD to node u
End AddRpca

Begin OnRecv_AddRpca
1    node u sends a broadcast message RPCA_ADDED
2    F = F ∪ {u}
End OnRecv_AddRpca
```

Figure B.4 *AddRpca* procedures in ARAM

SwitchRpca

Node *k* sends message *RPCA_SWITCH*, which contains the replica, and deletes its replica. When node *u* receives *RPCA_SWITCH*, node *u* sends a broadcast message *RPCA_SWITCHED* to inform all nodes that a new replica is added to node *u* and the replica at node *k* is deleted.

```
Begin SwitchRpca
  1   node k sends message RPCA_SWITCH to node u
  2   F = F ∪ {u} - {k}
End SwitchRpca

Begin OnRecv_SwitchRpca
  1   node u sends a broadcast message RPCA_SWITCHED
  2   F = F ∪ {u} - {k}
End OnRecv_SwitchRpca
```

Figure B.5 *SwitchRpca* procedures in ARAM

DeleteRpca

Node *k* sends a broadcast message *RPCA_DELETED* to inform all nodes in the system that its replica is deleted. Next, node *k* deletes the replica from its local cache.

```
Begin DeleteRpca
  1   node k sends a broadcast message RPCA_DELETED
  2   F = F - {k}
End DeleteRpca

Begin OnRecv_DeleteRpca
  1   F = F - {k}
End OnRecv_DeleteRpca
```

Figure B.6 *DeleteRpca* procedures in ARAM

ReadRpca

If node *k* has the replica in its local cache, node *k* increase the counter of local reads *R(k, k)* by 1 and the read operation succeeds locally. Otherwise, node *k* sends a message *RPCA_READ* to a replica node *n* which has the lowest weight value. When it receives the *RPCA_READ* message from node *k*, node *n* updates its read list information and replies with message *RPCA_READ_REPLY.*

```
Begin ReadRpca
1   if node k ∈ F then
2      R(k, k) → R(k, k) + 1
3      return O
4   else
5      q(n) → min q(r) and r∈C(k)
6      node k sends message RPCA_READ to node n
7      if node k receives message RPCA_READ_REPLY then
8         return O
9      end if
10  end if
End ReadRpca

Begin OnRecv_ReadRpca
1 if  k ∉ R then
2     R(n, k) = 1
3 else
4     R(n, k) → R(n, k) + 1
5 end if
5  node n sends message RPCA_READ_REPLY to node k
End OnRecv_ReadRpca
```

Figure B.7 ReadRpca and OnRecv_ReadRpca procedures in ARAM

UpdateRpca

If node *k* has the replica in its local cache, node *k* increases the counter of local write *W(k, k)* by 1 and updates the replica. Next, node *k* sends *RPCA_FORWARD* to a replica node *n* which has the lowest weight value. When node *n* receives the *RPCA_FORWARD* message from node *k*, if *k* is not in its write list *W*, it inserts *k* into *W* and set *W(n, k)* = 1. If *k* is already in the write list *W*, it updates *W(n, k)* by 1. Then, node *n* sends *RPCA_UPDATE* to other replica nodes. When a replica node *j* receives message *RPCA_UPDATE* from node *n*, it updates its write list. Finally, node *j* updates its replica.

```
Begin   UpdateRpca
1       if    node k ∈ F   then
2             W(k, k) → W(k, k) + 1
3             update replica O
4       end if
5       q(n) → min q(r) and r∈C(k)
6       node k sends message RPCA_FORWARD to node n
End UpdateRpca

Begin   OnRecv_ForwardRpca
1       if   k ∉ W then
2            W(n, k) = 1
3       else
4            W(n, k) → W(n, k) + 1
5       end if
6       update replica O
7       for  each node j ∈ F and j≠ k  do
8            node n sends message RPCA_UPDATE to node j
9       end for
End OnRecv_ForwardRpca

Begin OnRecv_UpdateRpca
1       if   n ∉ W   then
2            W(j, n) = 1
3       else
4            W(j, n) → W(j, n) + 1
5       end if
6       update replica O
End   OnRecv_UpdateRpca
```

Figure B.8 UpdateRpca procedures in ARAM

APPENDIX C - MFDRM ALGORITHM

This section lists the details and pseudo code of the MFDRM algorithm. Figure C.1 lists the variables and constants in MFDRM. Figure C.2 lists different types of message types in MFDRM.

Variable/Constant	Definition
k	the replica node that executes the algorithm MFDRM
$d(k, j)$	the shortest hop distance between node k and node j
R_{in}	the number of reads from node k itself during a time period T_n
RL	a list whose entry represents the number of reads that node k receives. For example, *RL(j)* means the number of reads, which are sent from node j, receives by node k
WL	a list whose entry represents the number of writes that node k receives. For example, *WL(j)* means the number of reads, which are sent from node j, receives by node k
W_{in}	the number of writes from node k itself during a time period T_n
F	the set of replica nodes in the systems
T_n	the time duration of the n^{th} replica allocation interval
A_n	the total number of data access during the time period T_n
N	the number of *RPCA_ADD_PERMIT* messages that replica node k receives
D	the number of *RPCA_DELETE_PERMIT* messages that replica node k receives
V	the set of all mobile nodes in the system
state	a variable indicates the state of node k
MAX_INTERVAL	the maximum replica allocation interval
MIN_INTERVAL	the minimum replica allocation interval
MAX	the maximum number of replica
MIN	the minimum number of replica

Figure C.1 Variables and constants in MFDRM

Message Type	Message Functionality
RPCA_READ	query a replica
RPCA_READ_REPLY	a query response
RPCA_WRITE	a replica update message
RPCA_ADDING	coordinate the adding of a new replica to the system
RPCA_ADD_PERMIT	the permission of adding a replica from a single node
RPCA_ADDED	add a replica to the system
RPCA_ADDED_REPLY	a broadcast message that replica has been added to the system
RPCA_DELETING	coordinate the deleting of a replica from the system
RPCA_DELETE_PERMIT	the permission of deleting a replica from a single node
RPCA_DELETED	a broadcast message indicating that the replica is deleted
RPCA_SWITCH	switch a replica in the system
RPCA_SWITCH_REPLY	a broadcast message indicating a replica is switched in the system

Figure C.2 Messages in MFDRM

AllocateRpca

Figure C.3 shows pseudo code of *AllocateRpca*. Node *k* first executes *AddRpca* procedures and *DeleteRpca* procedures, as in lines 1 and 2. Next, if both add and/or delete a replica fail and the current number of replica reaches *MAX*, node *k* applies the migration process. Finally, node *k* executes procedure *NextInterval* to set the next replica allocation interval.

```
Begin AllocateRpca
1   execute AddRpca procedure
2   execute DeleteRpca procedure
3   if AddRpca and DeleteRpca fails and |F| = MAX then
4       for each i ∈ RL do
5           if RL[i] > R_in then
6               execute SwitchRpca
7               break;
8           end if
9       end for
10  end if
11  execute NextInterval procedure
End AllocateRpca
```

Figure C.3 *AllocateRpca* procedure in MFDRM

AddRpca

If the current number of replicas in the system is less than the maximum number of replicas in the system, the add procedure starts. Replica node *k* search the *RL* list to find the first node *u* that satisfies (3.10) as in line 3 and node *k* changes its state from *Start* to *Adding* as in line 4. Next, replica node *k* sends coordination messages *RPCA_ADDING* to other replica nodes to request permission to add a replica. A replica node sends a permission message *RPCA_ADD_PERMIT* back to node *k* only when its state is *Adding*. We denote the number of permission messages that replica node *k* receives while in the *Adding* state as *N* and the current number of replicas as */F/*. If $N + |F| < MAX$, which means replica node *k* has enough permissions to add a new replica, replica node *k* changes its state from *Adding* to *Added*. Otherwise, it changes its state from *Adding* to *Start*. When a replica node *k* is in the *Added* state, it adds a replica to node *u* by sending message *RPCA_ADD*. When node *u* receives the new replica, it sends a broadcast message *RPCA_ADD_REPLY* to inform all nodes in the system that a new replica is added. Finally, node *k* changes its state from *Added* to *Start* when it receives message *RPCA_ADD_REPLY*.

```
Begin AddRpca
1   If |F| < MAX then
2      for  each node u in RL   do
3           If  u satisfies 3.10  then
4                state → Adding
5                for  each node n, n∈F and n≠ k  do
6                     node k sends message RPCA_ADDING to node n
7                end for
8                break;
9           end if
10     end for
11     N → 0
12     for each received RPCA_ADDING_PERMIT do
13          N → N + 1
14     end for
15     If N + |F| < Max  then
19          state → Added
20          node k sends message RPCA_ADDED to node u
21     else
22          state → Start
23     end if
24     on received RPCA_ADDED_REPLY do
25          F = F ∪ {u}
26     end do
27   end if
End AddRpca

Begin OnRecv_RpcaAdd
1      F = F ∪ {u}
2      node u sends broadcast message RPCA_ADDED_REPLY
End OnRecv_RpcaAdd
```

Figure C.4 *AddRpca* procedures in MFDRM

DeleteRpca

If node *k* satisfies 3.11, it changes its states from *Normal* to *Deleting.* Next, it sends coordination message *RPCA_DELETING* to other replica nodes to request permission to delete its replica, as in line 4. A replica node sends a permission message back to node *k* only when its state is *Deleting.* We denote the number of permission messages that replica node k in *Deleting* state receives as *D* and the current number of replica as |F|. If */F| - MIN > D*, which means replica node *k* has enough permissions to delete its replica, replica node k changes its state from *Deleting* to *Deleted,* as in line 12. When a replica

node *k* is in the *Deleted* state, it deletes its replica and sends message *RPCA_DELETED* to inform all nodes in the system that its replica is deleted. Finally, it changes its state from *Deleted* to *Start*.

```
Begin DeleteRpca
1   If   node k satisfies 3.11   then
2        state → deleting
3        for  each node n, n∈F and n≠k   do
4             node k sends message RPCA_DELETING to node n
5        end for
6        D → 0
7        for  each received message RPCA_DELETE_PERMIT   do
8             D → D + 1
9        end for
10       if   D < |F| - MIN   then
11            F = F ∪ - {k}
12            state → Deleted
13            node k broadcasts message RPCA_DELETED
14            isDeleted → true
15       end if
16       state  → Normal
12  end if
End DeleteRpca


Begin OnRecv_RpcaDeleted
1   F = F ∪ - {k}
End OnRecv_RpcaDeleted
```

Figure C.5 *DeleteRpca* procedure in MFDRM

SwitchRpca

Node *k* searches the *outRead* list to find the first node *u* that satisfies (3.4) and changes its state from *Start* to *Switch*. When replica node *k* is in the *Switch* state, it sends a message *RPCA_SWITCH* to node *j*. Upon receiving *RPCA_SWITCH* from replica node *k*, node *j* adds a replica to its local cache and sends a broadcast message *RPCA_SWITCH_REPLY* to all nodes in the system to indicate the switching of replica from node *k* to node *j*. Finally, when node *j* receives *RPCA_SWITCH_REPLY*, it changes its state from *Switch* to *Start*.

```
Begin SwitchRpca
1    for  each node i in RL do
3         if  RL(i) > R_in  then
4              state  → Switch
5              node k sends message RPCA_SWITCH to node u
6              break;
7         end if
8    end for
9     On receive RPCA_SWITCH_REPLY do
10        F = F ∪ - {k} + {u}
11        state → Start
End SwitchRpca

Begin OnRecv_RpcaSwitch
1    node u sends broadcast message RPCA_SWITCH_REPLY
2    F = F ∪ - {k} + {u}
End OnRecv_RpcaSwitch
```

Figure C.6 *SwitchRpca* procedures in MFDRM

ReadRpca

If node *k* has the replica in it local cache, node *k* increases the counter of r_{in} by 1 and the read operation succeeds locally. Otherwise, node *k* sends a message *RPCA_READ* to a replica node *n* which has the fewest hops to node *k*. When receives the *RPCA_READ* message from node *k*, node *n* either inserts *k* into the list and set the counter to 1 or increases the counter by 1. The read operation succeeds when node *k* receives the message *RPCA_READ_REPLY* from node *n*.

```
Begin ReadRpca
1   if node k ∈ F then
2       r_in → r_in + 1
3       return O
4   else
5       n → min_{j∈F} d(k, j)
6       node k sends message RPCA_READ to node n
7       if node k receives message RPCA_READ_REPLY then
8          return O
9       end if
10  end if
End ReadRpca

Begin OnRecv_ReadRpca
1 if node k ∉ RL then
2       RL(k) = 1
3 else
4       RL(k) → RL(k) + 1
5 end if
6 node n moves k to the first entry in RL
7 node n sends message RPCA_READ_REPLY to node k
End OnRecv_ReadRpca
```

Figure C.7 *ReadRpca* procedures in MFDRM

UpdateRpca

If node *k* has the replica in its local cache, node *k* increases the counter of W_{in} by 1 and updates the replica. Next, node *k* sends *RPCA_WRITE* to other replica nodes. When a replica node *n* receives the *RPCA_WRITE* message from node *k*, if *k* is not in *WL*, it inserts *k* into WL and set *WL(k)* equals 1. If *k* is already in *WL*, it increases its counter *WL(k)* by 1 and updates its replica.

```
Begin UpdateRpca
1   if node k ∈ F then
2       w_in → w_in + 1
3       update replica O
4   end if
5   for each node n ∈ F and n ≠ k do
6       node k sends message RPCA_WRITE to node n
7   end for
End UpdateRpca

Begin OnRecv_UpdateRpca
1   if node k ∈ WL then
2       insert k into WL and WL[K] → 1
3   else
4       WL[k] → WL[k] + 1
5   end if
6    update replica O
End OnRecv_UpdateRpca
```

Figure C.8 *UpdateRpca* procedures in MFDRM