Numéro d'ordre : 08/2007-D/IN

République algérienne démocratique et populaire
Ministère de l'enseignement Supérieure et de la recherche scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique, Département d'Informatique

Thèse
Présentée pour l'obtention du diplôme de Doctorat
En : Informatique
Spécialité : Programmation et Systèmes
Par : DERHAB Abdelouahid

Sujet :

Protocoles localisés pour la réplication de données et de services dans les réseaux mobiles ad-hoc

Soutenue le 15/10/2007, devant le jury composé de:

Mr.	M. AHMED-NACER	Professeur, USTHB	Président
Mr.	N. BADACHE	Professeur, USTHB	Directeur de thèse
M^{me}.	A. MOKHTARI	Professeur, USTHB	Examinateur
Mr.	A. BOUABDALLAH	Professeur, UTC (France)	Examinateur
Mr.	M. BOUFAIDA	Professeur, UMC	Examinateur
Mr.	O. NOUALI	Maitre de recherche, CERIST	Examinateur

Localized Protocols for Data and Service Replication in Mobile Ad-hoc Networks

Abstract

An ad-hoc network is a collection of mobile nodes forming a temporary network without any form of centralized administration or predefined infrastructure. In such a network, nodes move freely and their batteries drain out quickly. These lead to frequent network partitions, which may significantly degrade data and service availability. In such circumstances, replicating data or services at multiple nodes may improve data availability and response time.

In this thesis[1], we propose six localized replication protocols for mobile ad-hoc networks, where each node can make decision based only on the information from nodes within a constant hop distance. Network partitioning, energy consumption, and scalability are the three major issues that are considered in the design of these protocols. We first propose two partition prediction algorithms, the first one is for a single topology change and the second one is for concurrent topology changes. The algorithms can determine the time at which network partitioning might occur and replicate data items and services beforehand. We then propose three location-based data replication protocols that can achieve a good balance between scalability and availability. The last protocol we propose is based on clustering approach, where each node can send update and query messages to a cluster-head node that is within a constant hop distance.

Our simulation results and analytical studies show that the proposed replication protocols experience low cost, high data and service availability, and high data accuracy.

Keywords: localized replication, location-based protocol, partition prediction algorithm, availability, accuracy, scalability.

[1]This work has been carried out at CERIST research center and supported by Ministry of Higher Education and Scientific Research

Acknowledgements

I would like to thank my advisor, Prof. Nadjib Badache, for his guidance and support through the long march of my studies and research, and for providing me with the opportunity to attain this degree.

I would also like to thank Professors Mohamed Ahmed-Nacer, Aicha Mokhtari, Abdelmadjid Bouabdallah, Mahmoud Boufaida, and Doctor Omar Nouali for agreeing to serve on my thesis committee.

Visiting University of Compiegne was an educating experience for me. For that I thank Professor Abdelmadjid Bouabdallah who made my visits possible, and was always willing to contribute his valuable advice to my research.

Many thanks to all the colleagues and friends with whom I shared a laboratory, especially the head of my laboratory at CERIST research center, Mrs. Hassina Aliane for the support and help she has provided.

I would like to take this opportunity to thank my office-mates, both past and present, for providing me with a friendly and enjoyable work environment. My officemates included: Djamel Djenouri, Lyes Khelladi, Mehdi Chelbabi, and Hichem Abdellah-Hadj.

Finally, I am immensely indebted to my parents, my brothers, and my sisters for their support throughout my everlasting studies.

Table of Contents

List of Figures

List of Tables

Introduction

Wireless Communication between mobile users is becoming more popular than ever before. This is due to the recent technological advances in laptop computers and wireless data communication devices, such as wireless modems and wireless LANs. This has lead to lower prices and higher data rate, which are the two main reasons why mobile computing continues to enjoy rapid growth.

There are two distinct approaches for enabling wireless communication between two hosts. The first approach is to let the existing cellular network infrastructure carry data as well as voice. The major problems include the problem of handoff, which tries to handle the situation when a connection should be smoothly handed over from one base station to another base station without noticeable delay or packet loss. Another problem is that networks based on the cellular infrastructure are limited to places where there exists such a cellular network infrastructure.

The second approach is to form a Mobile Ad-hoc NETwork (MANET) among all users wanting to communicate with each other. An ad-hoc network is a collection of autonomous wireless nodes that may move unpredictably, forming a temporary network without any fixed backbone infrastructure. This means that all users participating in the ad-hoc network must be willing to forward data packets to make sure that the packets are delivered from source to destination. This form of networking is limited in range by the individual nodes transmission ranges and is typically smaller compared to the range of cellular systems. Ad-hoc networks are useful in places with damaged communication infrastructure where rapid deployment of a communication is needed. They can also be useful in situations, in which users want to form a temporary network.

In ad-hoc networks, since mobile nodes move freely and and run out of battery power so quickly, disconnections may occur frequently. If a network is divided into multiple partitions, mobile nodes in one of the partitions cannot access the data or

services held by the other partitions. To deal with these issues, data and services can be replicated at multiple nodes to improve data availability. Further, data replication can also reduce the query delay, since mobile nodes can get the data from some nearby replicas.

The goal of this thesis is to provide novel solutions for data and service replication in mobile ad-hoc networks. The solutions are localized, i.e. nodes make decisions based solely on the knowledge of their k-hop neighborhood. They take into consideration the issues arising from constraints imposed by the ad hoc environment, such as: network partitioning, energy limitation, and scalability. They also try to make tradeoffs between data availability, data accuracy, and cost metrics.

The rest of the thesis is organized as follows: Chapter 1 introduces the mobile ad-hoc networks and its characteristics. It also explains the difficulties encountered to implement services and protocols in such networks. Chapter 2 defines some terminologies that will be later used in this thesis, presents the fundamental design issues to be considered when developing a replication protocol for MANETs, and proposes a classification scheme that categorizes the replication protocols into various classes, with respect to the issues they address. Chapter 3 presents and compares the existing replication protocols. Chapter 4 proposes a partition-aware replication protocol. The protocol can determine the time at which network partitioning might occur and replicate data items and services beforehand. However, the proposed protocol can only tolerate a single topology change. To fix this shortcoming, a self-stabilizing partition-aware replication protocol that can tolerate concurrent topological changes is proposed in Chapter 5. Chapter 6 proposes three scalable location-based data replication protocols. Simulation results and analytical studies show that the location-based replication protocol can achieve a good balance between scalability and availability. Chapter 7 proposes another scalable data replication protocol, it is cluster-based and aims at making a tradeoff between availability, consistency and update cost.

Chapter 1

Mobile ad-hoc networks

1.1 Introduction

Mobile ad hoc networks [116, 16] are formed dynamically by an autonomous system of mobile nodes that are connected via wireless links without using an existing network infrastructure or centralized administration. The nodes are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a standalone fashion, or may be connected to the larger Internet. Mobile ad hoc networks are infrastructureless networks since they do not require any fixed infrastructure such as a base station for their operation. In general, routes between nodes in an ad hoc network may include multiple hops and, hence, it is appropriate to call such networks "multi-hop wireless ad hoc networks".

Figure 1.1 shows an example of mobile ad hoc network and its communication topology. As shown in Figure 1.1, an ad hoc network might consist of several home-computing devices, including notebooks, handheld PCs, and so on. Each node will be able to communicate directly with other nodes that reside within its transmission range. For communicating with nodes that reside beyond this range, the node needs to use intermediate nodes to relay messages hop by hop.

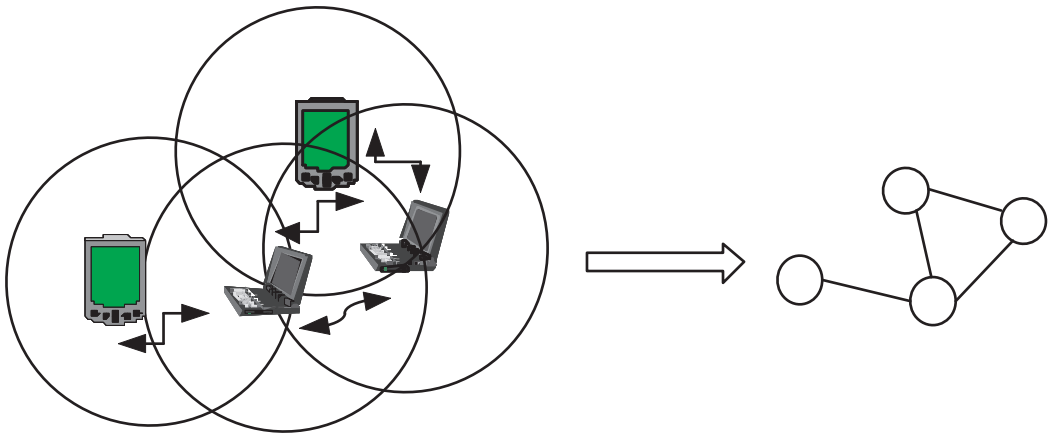


Figure 1.1: Mobile ad hoc network

1.2 Characteristics and advantages of mobile ad-hoc networks

MANETs inherit common characteristics found in wireless networks in general, and add characteristics specific to ad hoc networking:

- *Wireless:* Nodes communicate wirelessly and share the same media (radio, infrared, etc.).
- *Mobility:* Each node is free to move about while communicating with other nodes. The topology of such an ad hoc network is dynamic in nature due to constant movement of the participating nodes.
- *Multi-hop routing:* No dedicated routers are necessary; every node acts as a router and forwards each others' packets to enable information sharing between mobile hosts.
- *Autonomous and infrastructureless:* MANET does not depend on any established infrastructure or centralized administration. Each node operates in distributed peer-to-peer mode, acts as an independent router.

1.3 MANET Applications

Historically, mobile ad hoc networks have primarily been used for tactical network-related applications to improve battlefield communications and survivability. The dynamic nature of military operations means it is not possible to rely on access to a fixed communication infrastructure on the battlefield.

Although early MANET applications and deployments were military oriented, non-military applications have grown substantially since then and have become the main focus today. Especially in the last few years, with the rapid advances in mobile ad hoc networking research, mobile ad hoc networks have attracted considerable attention and interest from the commercial sector as well as the standards community. The introduction of new technologies such as Bluetooth, IEEE 802.11, and Hyperlan greatly facilitate the deployment of ad hoc technology outside of the military domain. As a result, many new ad hoc networking applications have since been conceived to help enable new commercial and personal communications beyond the tactical networks domain, including personal area networking, home networking, law enforcement operations, search-and-rescue operations, commercial and educational applications, sharing information in a conference, sensor networks, and so on.

1.4 Design Issues and Constraints

As described in the previous section, the ad hoc architecture has many benefits, such as self-reconfiguration, ease of deployment, and so on. However, this flexibility and convenience create a number of complexities and design constraints that are new to mobile ad hoc networks, which are:

- *Lack of infrastructure:* The lack of a fixed infrastructure and a centralized entity mean that network management has to be distributed across different nodes, which brings added difficulty in fault detection and management.

- *Dynamic network topology:* In mobile ad hoc networks, since nodes can move arbitrarily, the network topology, which is typically multi-hop, can change frequently and unpredictably, resulting in route changes, frequent network partitions, and possibly packet losses.

- *Physical layer limitation:* The radio interface at each node uses broadcasting for transmitting traffic and usually has limited wireless transmission range, resulting in specific mobile ad hoc network problems like hidden terminal problems, exposed terminal problem, and so on. Collisions are inherent to the medium, and there is a higher probability of packet losses due to transmission errors compared to wireline systems.

- *Limited link bandwidth:* Because mobile nodes communicate with each other via bandwidth-constrained, variable capacity, and error-prone wireless channels, wireless links will continue to have significantly lower capacity than wired links and, hence, congestion is more problematic.

- *Variation in link and node Capabilities:* Each node may be equipped with one or more radio interfaces that have varying transmission/receiving capabilities. This heterogeneity in node radio capabilities can result in possibly asymmetric links, and designing network protocols and algorithms for this heterogeneous network can be complex.

- *Energy constrained operation:* Because batteries carried by each mobile node have limited power, processing power is limited, which in turn limits the lifetime of services and applications that can be supported by each node. If some nodes die due to the lack of energy [98], it may result in a lack of connectivity between nodes that are alive. Turning off network devices to conserve energy may also lead to network partitioning [111].

- *Network security:* Mobile wireless networks are generally more vulnerable to information and physical security threats than fixed-wireline networks [107]. The use of open and shared broadcast wireless channels means nodes with inadequate physical protection are prone to security threats. The security requirements in ad-hoc networks include: preventing eavesdropping, protecting access to wireless network infrastructure, preventing tampering with traffic (i.e., accessing, modifying or injecting traffic), and protection against denial of service attacks by malicious nodes.

- *Network scalability:* Current popular network management algorithms were mostly designed to work on fixed or relatively small wireless networks. Many

mobile ad hoc network applications involve large networks with tens of thousands of nodes, scalability is critical to the successful deployment of such networks. The evolution toward a large network consisting of nodes with limited resources is not straightforward and presents many challenges that are still to be solved.

1.5 Media Access Control in ad-hoc networks

In MANET, use of broadcasting and shared transmission media introduces a nonnegligible probability of packet collisions and media contention, which severely reduces channel utilization as well as throughput, and brings new challenges to conventional CSMA/CD-based and MAC protocols in general. Among the top issues are the hidden-terminal and exposed-terminal problems.

The hidden-terminal problem occurs when two (or more) terminals, say, A and C, cannot detect each other's transmissions (due to being outside of each other transmission range) but their transmission ranges are not disjoint. As shown in Figure 1.2, a collision may occur, for example, when terminal A and C start transmitting toward the same receiver, terminal B in the figure.

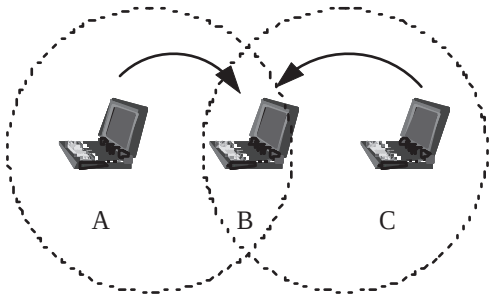


Figure 1.2: Hidden-terminal problem

The exposed-terminal problem results from situations in which a permissible transmission from a mobile station (sender) to another station has to be delayed due to the irrelevant transmission activity between two other mobile stations within sender's transmission range. Figure 1.3 depicts a typical scenario in which the exposed-terminal problem may occur. Let us assume that terminals A and C can hear transmissions from B, but terminal A cannot hear transmissions from C. Let us also assume that terminal B is transmitting to terminal A, and terminal C has a frame to

be transmitted to D. According to the CSMA scheme, C senses the medium and finds it busy because of B's transmission, and, therefore, refrains from transmitting to D, although this transmission would not cause a collision at A. The exposed-terminal problem may thus result in loss of throughput.

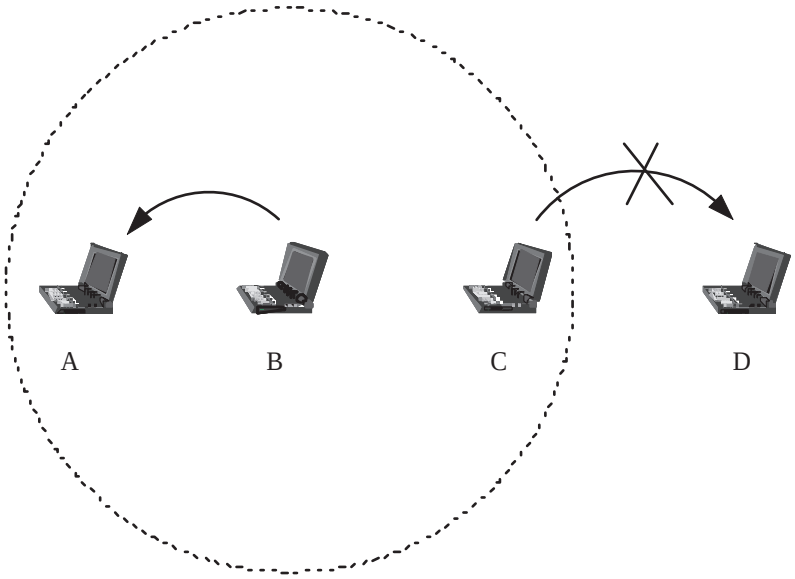


Figure 1.3: Exposed-terminal problem

A very large number of new-generation ad hoc protocols such as MACA (multiple access with collision avoidance protocol), MACAW (MACA with CW optimization), FAMA (floor acquisition multiple access), MACA/PR and MACA-BI (multiple access with collision avoidance by invitation protocol) [80, 19, 49, 94, 133] have been proposed to resolve the various hidden-terminal, exposed-terminal and similar problems, and improve channel performance in MANET. The key ideas behind these protocols involve sending RTS (request to send) and CTS (clear to send) packets before the data transmission has actually taken place. Specifically, before transmitting a data frame, the source station sends a short control frame, named RTS, to the receiving station, announcing the upcoming frame transmission. Upon receiving the RTS frame, the destination station replies by a CTS frame to indicate that it is ready to receive the data frame. Both the RTS and CTS frames contain the total duration of the transmission, that is, the overall time interval needed to transmit the data frame and the related ACK. This information can be read by any station within the transmission range of either the source or the destination station. Hence, stations become aware of

transmissions from hidden stations, and the length of time the channel will be used for these transmissions.

However, studies [62, 34] show that when traffic is heavy, a data packet can still experience collision due to loss/collision of RTS or CTS packets. To alleviate this problem, comprehensive collision-avoidance mechanisms have been introduced via a backoff mechanism. In principle, once a transmitting node senses an idle channel, it waits for a random backoff duration (determined by a contention window, and increasing exponentially with each reattempt) before attempting to transmit the packet, and congestion control is achieved by dynamically choosing the contention window based on the traffic congestion situation in the network.

1.6 Ad-hoc routing

In the absence of a fixed infrastructure, nodes have to cooperate in order to provide the necessary network functionality. Routing is one of the primary functions each node has to perform in order to enable connections between nodes that are not directly within each others transmission range. The development of efficient routing protocols in ad-hoc networks is a nontrivial and challenging task because of the high dynamic and unpredictable topology of such networks.

In the literature, we distinguish two different approaches: topology-based and position-based routing. Topology-based routing protocols use the information about the links that exist in the network to perform packet forwarding. They can be further divided into proactive [132, 114], reactive [115, 74], and hybrid [56] approaches.

Proactive protocols maintain routing information about the available paths in the network even if these paths are not currently used. The main drawback of these approaches is that the maintenance of unused paths may occupy a significant part of the available bandwidth if the topology of the network changes frequently.

Reactive protocols, on the other hand, maintain only the routes that are currently in use, thereby reducing the burden on the network when only a small subset of all available routes is in use at any time. However, they still have some limitations. First, since routes are only maintained while in use, it is typically required to perform a route discovery before packets can be exchanged between communication peers. This leads to a delay for the first packet to be transmitted. Second, even though route maintenance for reactive algorithms is restricted to the routes currently in use, it may

still generate a significant amount of network traffic when the topology of the network changes frequently. Finally, packets on route to the destination are likely to be lost if the route to the destination changes.

Hybrid ad hoc routing protocols combine local proactive routing and global reactive routing in order achieve a higher level of efficiency and scalability. In the Zone Routing Protocol (ZRP) [56], a route discovery is initiated on demand. A routing zone is defined for each node and includes the nodes whose distance is less than a predetermined maximum number of hops. Each node is required to know the topology of the network within its zone only. Updates about changes in topology within the zone are propagated by using a proactive routing protocol. Each node therefore, has a route to all other nodes in the same zone. If the destination node resides outside the source zone, a reactive search-query routing method is used. The disadvantage of ZRP is that for large values of routing zone the protocol can behave like a pure proactive protocol, while for small values it behaves like a reactive protocol.

To eliminate some of the limitations of topology-based routing, the position-based routing [15, 84, 81, 73, 93, 131] is proposed. The latter uses the geographic position of nodes available from positioning systems such as GPS [79] or other type of positioning service [5, 22, 6] to forward data packets. The position-based routing protocols have several advantages. First, the forwarding decision at each node is based on the destinations position and the position of the neighboring nodes. Typically, the packet is forwarded to a neighbor that is closer to the destination than the forwarding node itself, thus making progress toward the destination. In order to inform all neighbors within transmission range about its own position, a node transmits beacons at regular intervals. These protocols are localized since the knowledge of each node is limited to one hop. Second, they consume less overhead as they do not require to establish or maintain routes. Third, they can scale to a large number of nodes, since nodes do not have global knowledge of the identities of other nodes in the network. To enable position-based routing, a node must be able to discover the location of the node whom it wants to communicate with. Location information are provided by a so-called location service. The role of a location service is to map the ID of a node to its geographical position. Each location service performs two basic operations: the *location update* and the *location query*. The location update is responsible for replicating information about the current location of a given node D to a set of nodes called *location servers*. If a node S wants to know the location of node D, it sends

a location query message to some or all the location servers of node D. The only drawback of the current greedy approaches is that the position of the destination needs to be known with an accuracy of a one-hop transmission range; otherwise, the packets cannot be delivered.

The comparison between different routing protocols was the subject of many works [13, 12, 43, 21, 31, 23, 29, 127].

1.7 Fault-tolerant algorithms in ad hoc networks

Several coordination problems such as: distributed mutual exclusion, consensus, leader election, and group communication, have been well-studied in wired networks. These problems are considered as important building blocks in distributed systems [97, 8, 75, 14].

In particular, certain applications of ad hoc networks require primarily distributed services to coordinate the collective actions of nodes. We can find many applications that have immediate needs for both ad hoc networking and distributed coordination services. For example, using teamed robots for unmanned explorations and rescue operations becomes an increasingly tempting application. Therefore, several ongoing researches [76] are focussing on coordinating robots with wireless communication networks. Another example [121] is a set of vehicles running through critical points, such as highway entrances and blind crossings (crossings without light control), have to be scheduled to share the resources (i.e., those critical points), in order to avoid collisions. Distributed algorithms relying on inter-vehicle communications could be a conflict resolution method performed by the vehicles themselves.

Recently, there has been considerable interest in using leader election [100, 63, 139, 110], mutual exclusion [141, 140, 27, 37], and group and routing coordination [123, 89, 7, 124, 103, 122, 69, 50] algorithms in ad hoc networks.

By their nature, network applications for mobile computing involve cooperation among multiple sites. For these applications, which are characterized by reliability and reconfigurability requirements, possible partitioning of the communication network is an extremely important aspect of the environment. In addition to accidental partitioning caused by failures and node movement, mobile computing systems typically support disconnected operations, i.e., a mobile host may intentionally decide to disconnect itself from the network and work only locally, which is an additional cause

of partitioning. Intuitively, partitions correspond to maximal connected components of the logical graph representing the reachable relation among nodes. Partitioning may result in service degradation but need not necessarily render services completely unavailable.

Informally, we can define the class of partition-aware applications as those that are able to make progress in multiple concurrent partitions without blocking. Service reduction and degradation depend heavily on the application semantics. For certain application classes with strong consistency requirements, it may be the case that all services have to be suspended completely in all but one partition. This situation corresponds to the so-called primary component model.

Babaoglu et al. [11] present an example of partition-aware applications, which can build upon a partitionable group membership service, called *Partitionable Service Activator*. They consider a network service for distributing a continuous stream of data (e.g., audio, video, stock quotes, news headlines) to a collection of subscribers. The data distribution can be provided by any one of a set of servers that have access to the data source. The service should be available in every partition that contains at least one server; furthermore, to minimize resource usage, multiple active servers within the same partition should be avoided. New servers may be added and existing ones removed at will by an administrator. The goal is to devise a service activator algorithm such that a server can decide when it should be active and when it should be passive. A solution must activate a new server if the current one is removed from the system, if it crashes or if it ends in another partition.

Such distributed, partitionable applications are well suited for ad hoc networks due to their peer-to-peer architecture. However, important network applications and services such as web servers, location information databases, and network services are inherently centralized [142]. These services are often critical to the mobile node's operation such that every node requires constant and guaranteed access to them. When the network partitions, those mobile users that are not in the same partition as the centralized server lose access to the service. To ensure that the service is available to all nodes, a trivial solution is to place the service on every mobile node, so that the service availability is independent of any changes in the network topology. However, this trivial solution incurs a prohibitively high service cost (in terms of the number of servers deployed). Several research works [142, 26, 25] have addressed the problem of maintaining the network-wide coverage of the centralized service in the

presence of frequent partitioning, and without incurring high service cost.

Current fault-tolerant algorithms for partitionable mobile ad-hoc networks assume either that: (a) mobile nodes move in a free space with bounded physical speed and never fail, or that: (b) the network re-connects infinitely often so that the algorithm can make progress. Although in a real system (b) may hold for most of the time (i.e., with high probability), it can still be the case that waiting for network components to re-merge may require unbounded node resources and communication delays. Therefore, we retain that in practice the problem of permanent partitioning cannot be avoided.

1.8 Conclusion

In this chapter, we have presented the ad hoc mobile networks, its characteristics, advantages, limitations, and its application fields. We have also presented a brief overview of MAC protocols in ad-hoc network. Then, we have summarized the existing classes of routing protocols, and explained how it is difficult to ensure routing between users in such networks. We have also explained how the network partitioning issue affects the behavior, and the performance of services and applications. In the next chapter, we will discuss the issues to be considered when designing a replication protocol for ad hoc networks, and based which, a taxonomy of the existing replication protocols is proposed.

Chapter 2

Replication protocols in Mobile Ad-hoc Networks: Issues and a taxonomy

2.1 Introduction

Accessing remote services and data is one of the most important applications in both fixed and mobile networks. As an example, a battlefield communication system where a platoon of soldiers and vehicles that form an ad-hoc network, move together to complete a task. Soldiers should be able to gather the latest information such as maps and share this information with others. In the same context, we can find various data sharing services: Mobility management [112, 55, 54], distributed management of cryptographic keys and certificates [155, 85, 72] and distributed addressing service [113, 138, 106].

In distributed systems, a single server may serve many clients and the heavy load on the server may cause the response time to be adversely affected. In such circumstances, replicating objects (e.g., service or data) may improve performance.

Replication is a fundamental technique used in distributed systems. It consists of storing the same data or service at multiple nodes. Data or services are often replicated to improve availability, reliability, fault-tolerance, and performance. Replication may also improve data and service availability when the server crashes. The other issue facing data replication is the correctness of data. Node mobility and node

failure may lead to network partitioning, where the network is split up into disjoint partitions, given rise to the possibility of data inconsistency.

2.2 System model and definitions

The system consists of a set of n mobile nodes. The database of interest, denoted by DB, is either centralized or distributed. In the centralized database systems, a single node called server holds the whole database. The database in this case may also be a set of configuration items used by a given service. In the distributed database systems, each node i has a data item D_i associated with it (i.e., D_i is only updated by node i), and $DB = D_1, \cdots, D_n$. Each node can perform two types of operation: *update* and *query*. When a data item D_i is updated, node i sends an *update* message to one or multiple nodes, called replica servers, to modify the value of its replicas. When a node wants to query the value of a given data item, it sends a *query* message to one or multiple nodes that hold a copy of this data item. A replicated object (i.e., *replica*) is a data item that is stored redundantly at multiple replica servers. We define data replication as a technique of creating and managing duplicate versions of data items.

2.3 Replication versus caching

There are two different mechanisms to establish copies of data at different nodes: replication and caching. There are a number of subtle differences between replication and caching.

The first difference between replication and caching concerns the mechanism used to establish copies of data. Replication is carried out by a separate process that copies the data items to target servers. Caching, on the other hand, is a by-product of query execution. A client node keeps all the used data items in its cache, if the cache is large enough. Replication can occur at servers even if no queries are processed by these servers, whereas the cache of a client is empty if no queries have been processed by that client. As a consequence, caching decisions need to be made by the query process while replication decisions can be made by a separate component that is established at every server and works independently of the query process.

Second, replicas are kept by the servers until they are explicitly deleted whereas copies of data are kept in a client's cache until they are replaced by copies of other and more interesting data using a replacement policy such as LRU (Least Recently Used) or until they are removed from the cache because of invalidation.

Third, propagation-based protocols are used to keep replicas of data consistent and accessible at servers at all times. For caching, on the other hand, maintains the consistency by using protocols that are based on invalidation and removes out-of-date copies.

Fourth, replication copies all or the majority of data items, since a large group of clients benefit from replication, and it is quite likely that most parts of the data items will be used by this group of clients. Caching, on the other hand, copies only individual data items to the clients, because it supports the queries of a single client or of a fairly small group of clients, and clients tend to be only interested in a small fraction of the data stored at a specific server.

Although many caching protocols [149, 108, 24, 144] have been proposed for ad-hoc networks, but we focus in this thesis on replication protocols because they can improve data availability more than the caching ones. This is due to the fact that the replication protocols trigger the replication process independently of client queries.

2.4 Design issues of data replication protocols for ad hoc networks

Data replication has been extensively studied in distributed systems, especially in wired networks [68, 30]. In such systems, nodes that hold database are more reliable and less likely to disconnect or fail. They more focus on replica placements so as to improve the query latency and the update cost. They do not consider data availability as a big concern, since the failure of links and nodes is infrequent and a small number of replica servers can provide high data availability. In ad hoc networks, mobile nodes move freely and their batteries drain out quickly, which cause frequent link breakages and node failures. The failure of some links and nodes considered as critical can split up the network into several disjoint partitions. This situation considerably reduces data availability and gives rise to more data inconsistency. In addition, the shared nature of the wireless channel increases the cost of a remote access, since each 1-hop

query transmission must contend with other transmissions to access the channel. By replicating data at multiple nodes, data availability can be improved. Further, data replication can also reduce the query delay, since mobile nodes get the data from some nearby replicas.

In addition to availability and performance issues that have been well discussed in fixed networks, data replication in ad hoc networks must address additional issues arising from the constraints imposed by the ad-hoc network environment. These issues are the following:

- *Network partitioning issue:* Due to network partitioning in ad hoc networks, the chances to access a data item become low since the mobile users may be not in the same partition as the node holding the data item. Replicating data in future separate partitions before the occurrence of network partitioning can improve data availability. To do so, the replication protocol should determine the time at which network partitioning might occur and replicate data items beforehand.

- *Energy consumption issue:* Mobile nodes operate on low-power batteries. A single server may serve many clients, which leads that its battery is exhausted very quickly. To improve data availability, the replication protocol should replicate the critical data items on nodes that can last for a long time period. Moreover, it should also replicate data in such a way that the power consumption of servers is reduced and is balanced among all servers in the network.

- *Scalability issue:* As the network size increases, a query sent by a client node may need to traverse a long path to reach the server node, therefore increasing the query cost and latency. Moreover, the existence of a large number of querying nodes gives rise to more channel access contention among clients, which decreases considerably the available bandwidth and increases channel access delay. The replication protocol should be designed so that its performance is not greatly affected if the number of nodes or the network size increases.

When designing a data replication protocol, the following basic questions need to be considered. (1) When should a node trigger the replication process? (2) what criteria are used to select replica servers (i.e. servers that receive update messages)?

(3) how should a node find the appropriate servers to query for a data item, and (4) how can the protocol provide high data availability?

These questions highlight the need of designing a data replication protocol that addresses both availability and performance issues. To judge the merit of protocols, we use six performance metrics that can help promote meaningful comparisons and assessments of each protocol. These metrics are formally defined as follows:

- *Replication cost:* The number of replica servers (i.e. nodes holding data items).
- *Update cost:* The average number of hops each node needs to perform to carry out the update operation.
- *Query cost:* The average number of hops each node needs to perform to carry out the query operation.
- *Storage cost:* The number of data items a node needs to store as a replica server.
- *Data availability*[1]*:* If N_s denotes the number of successful attempts to access a data item, and N_a is the total number of attempts. The data availability is defined to be: $\frac{N_s}{N_a}$.
- *Data accuracy:* If N_q denotes the number of query operations. and N_o the number of outdated values returned by those queries. The data accuracy is defined to be: $\frac{(N_q - N_o)}{N_q}$.

In designing a data replication protocol for ad hoc networks, one must have clear goals, since optimizing all metrics is hard to achieve. In the following, we discuss how one metric is optimized at the expense of others.

- *Data availability vs. replication cost:* In ad-hoc networks, critical object may suffer from low availability in the face of of node failures, unreachable nodes, and attacks. Creating a large number of replicas can provide high availability. However, this strategy comes at the price of high cost, since each replica server needs to receive changes made by the original server so as to maintain up-to-date information. One of the challenges facing replication protocol is how to ensure high availability without generating a large number of servers.

[1]In the literature, data availability is also called data accessibility

- *Replication cost vs. query cost:* As the number of replicas increases, the number of hops traversed by a query request decreases, since client nodes are more likely to query near servers for the objects they need. However, decreasing the query cost increases the cost of updating replicas. The replication protocol should make a trade-off between update and query costs.

- *Update cost vs. accuracy:* Replication protocols differ in the way they update their replicas. In the first policy, a data item is propagated to the replica servers each time it is updated. Although this policy increases the consistency of local databases, but this comes at the price of an increase in update cost especially when data items are updated at high rate. In the second policy, the update cost is decreased by delaying the propagation of updates to the different replicas. However this policy increases the probability to access an invalid data item, and hence reduces the accuracy of information obtained from replica servers. The replication protocol should balance between data accuracy and update cost.

Motivated by these observations, a number of trade-offs must be made in designing a data replication protocol. The replication protocol needs to balance between *availability*, *query cost*, *update cost*, *storage cost*, and *accuracy*.

2.5 Classification of replication protocols

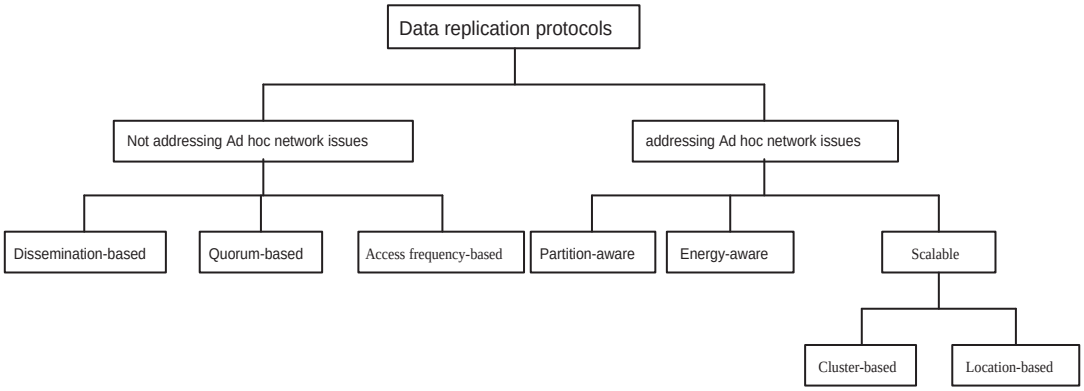


Figure 2.1: A taxonomy of replication protocols

Network partitioning, energy consumption, and scalability are the most important issues to be considered when designing a replication protocol for ad hoc networks. Based on these issues, we propose a taxonomy of the replication protocols as depicted in 2.1. At the top level, we divide the replication protocols into those that address and those that do not address the three criteria mentioned above.

The protocols that do not deal with the ad hoc network issues can be further divided into: dissemination-based, quorum-based, and access frequency-based approaches. In the dissemination-based approach, the update packet is flooded to all nodes in the network. In the quorum-based approach, the update packet is sent to a subset of nodes called *update quorum*, and the query packet is sent to a subset of nodes called *query quorum*. The subsets are designed such that each query quorum for a node intersects an update quorum for any other node. In the access frequency-based approach, each node maintains replicas of data that is frequently requested, and replication decisions are taken based on the access frequency to data items.

The protocols dealing with the ad-hoc network issues can be classified as *Partition-aware*, *Energy-aware*, and *Scalable* protocols. Partition-aware protocols try to predict network partitioning and replicate data items beforehand. Energy-aware protocols take into account the power consumption of client and server nodes. Scalable protocols try to design localized solutions, in which each node can decide on its own behavior based only on the information from all nodes within a constant hop distance. The scalable protocols can be further divided into location-based and cluster-based approaches. In the location-based approach, a node chooses its replica servers based on their positions and the area where they are located. In the cluster-based approach, all nodes are put into multiple non-overlapping groups and a cluster head is selected for each group to handle the control packets for replica query and update.

2.6 Conclusion

In this chapter, several design issues concerning the development of a replication protocol for mobile ad hoc networks have been discussed. Existing replication protocols have been classified based on the issues they address.

The design of an efficient replication protocol involves tradeoffs among six performance metrics, which are: replication cost, storage cost, update cost, query cost, data availability, and data accuracy. In the next chapter, we will provide overviews

of different replication protocols for ad-hoc mobile networks.

Any solution trying to deal with the problem of replication in ad-hoc networks has to take also into consideration the following features:

- When does the servers replicate data items, i.e., which parameters are considered for the decision to replicate ? Is it link stability, access frequency, energy consumption, or overhead?
- *Partition-aware:* Does the replication protocol predict network partitions before they occur and replicate data accordingly?
- *Energy-aware:* Does the replication protocol address node power limitations?
- *Scalability:* Is the replication protocol scalable ?
- *Network partitions prediction:* How does the protocol predict network partitions while taking the decision to replicate?
- What criteria are used to choose the mobile node that will hold the replicas?
- *Routing protocol dependency:* Does the replication protocol depend on any particular routing protocol?

Chapter 3

Replication protocols in Mobile Ad-hoc Networks: State of the art

3.1 Replication protocol not addressing the ad-hoc network issues

3.1.1 Dissemination-based replication protocols

In the dissemination approach, whenever a node updates its data item, it floods the update packet to all nodes in the network. Thus, when a given node requires the data item of another node, the information is found in the node's memory or in nearby nodes, i.e., the dissemination protocols usually do not send query packets.

In [64], the authors try to reduce the storage space by choosing the appropriate node to hold the replica. They propose a protocol, in which using only local information, each node upon receiving a disseminated data item, decides whether it caches a data item or not. For a given source node S holding the data item D_S, the first nodes that cache the data item must be located at a distance m to the source. Those nodes are at the first rank, called r_0. Each of those nodes will cover a zone located at a distance $(\lambda \times m)$ around them. They give a rule for constructing rank $(r+1)$ from rank r and probabilistic rule $P(d)$ so that a node u located at a distance d to the source S can decide by itself either it should cache the D_S or not. Each node u receiving the update packet, generates a random number p. If $p < P(u)$, it saves D_S in its cache.

Hayashi et al. [67] propose two updated data dissemination methods, called *DU* and *DC* to update old replicas. Each mobile node holds a table in which the information on the latest update times (timestamps) of all data items in the entire network is recorded.

In the *DU (Dissemination on Update)* method, when a mobile node updates its data item, it floods all connected mobile nodes with an invalidation report. When a mobile node receives the invalidation report, it checks whether the replica it holds is invalid. If so, it updates the time stamp in its own table to that in the received invalidation report, and broadcasts the received invalidation report to its neighboring nodes. If that node holds a replica of the corresponding data item, it discards the replica from its own cache and requests the updated data item to the node holding the original. When the node holding the original receives this request, it transmits the updated data item to the requesting node.

In the *DC (Dissemination on Connection)* method, a node holding an original data item floods other nodes with the invalidation report and disseminates the updated data item every time it updates the data item. In addition, every time two mobile nodes (M_i, M_j) are newly connected with each other, the flooding of invalidation reports is performed as follows:

1. The mobile node with the larger node identifier (M_j) sends its own table to the other one (M_i).

2. Node M_i compares the entry for each data item in its own table with that in the table received from M_j and updates its own table.

3. M_i floods invalidation reports for data items whose timestamps held by M_i are smaller than that held by M_j to mobile nodes originally connected to M_i.

4. M_i sends information on the updated time stamps for data items whose timestamps held by M_j are smaller than those held by M_i to M_j. Then, M_j floods the nodes originally connected to M_j with the invalidation reports for these items.

The traffic produced by the DC method is larger than that in the DU method since each mobile node floods others with invalidation reports and disseminates updated data items more frequently. To deal with issue, they have proposed three variants: *DC/One-to-One*, *DC/Group-to-Group*, and *DC/local group-to-local group* methods, which try to reduce the network traffic.

The dissemination approach is very costly in terms of storage space and message overhead. This approach does not address the issues of ad-hoc networks.

3.1.2 Quorum-based replication protocols

Karumanchi's protocol

Karumanchi et al. [82] have proposed a set of quorum-based strategies for information dissemination in mobile ad hoc networks with partitions. Given a set of S servers, structured into m fixed subsets (i.e. quorums) denoted by $S_0, S_1, \cdots, S_{m-1}$. Each node performs two operations: (1) information update and (2) information query. Four different policies that determine the time to send updates have been proposed as follows:

1. *The time-based strategy:* The time between successive update attempts by a node is exponentially distributed, with a mean of t units.

2. *The time and location-based strategy:* In this policy, a node remembers its location when it last sent an update. If the node's location unchanged since the last successful update, no updates are sent.

3. *The absolute connectivity-based strategy:* A node sends an update when a certain pre-specific number of links incident on it have been established or broken since the last update.

4. *The percentage connectivity-based strategy:* An update is triggered when a pre-specific percent of the links incident on it have changed since the last update.

When a node x wishes to update some information, it timestamps the datum with its local clock value. Then, the following actions are performed:

1. Node x randomly selects a quorum S_i from the set of quorums and sends *UPDATE* message, timestamped with its local clock value, to all servers in the quorum. Because of network partitioning, it may happen that, only $S_i' \subseteq S_i$ receive the *UPDATE* message.

2. The servers, on receiving the *UPDATE* message, overwrite their old copy of the data item with the new copy. If they do not have an old copy of that data

item, they simply add the information received in the message to their database. Optionally, the servers may also send a positive acknowledgment to x.

When a node y wants to perform the query operation, it executes the following actions:

1. Node y randomly selects a quorum S_j and sends a $QUERY$ message to all servers in the quorum. Because of network partitioning, it may happen that, only $S'_j \subseteq S_j$ receive the $QUERY$ message.

2. When a server receives a $QUERY$ for a datum and has a copy of it, the server sends a $REPLY$ containing the information along with the timestamp associated with the datum. Otherwise, the server sends a $NULL$ reply.

3. Receiving all the $REPLY$ messages, y selects the value of the datum with the largest timestamp.

There is a possibility that the query may not return any information or return a stale information, this is due to the fact that: $S'_i \cap S'_j = \phi$. To reduce $S_i - S'_i$ (increase the accuracy of information) and reduce $S_j - S'_j$ (increase the accessibility of information), the authors have defined the *Disqualified list*, DQL_x, which contains servers that x believes to be unreachable. Using the *Disqualified list*, they have also proposed three techniques, which are:

1. *Select_Then_Eliminate (STE) strategy*: Node x randomly selects quorum S_i and sends *query/update* to $(S_i - DQL_x)$. An update is considered to be successful if at least one server sends an ACK within a given timeout. In the case of queries, the received copy with the largest timestamp is selected.

2. *Eliminate_Then_Select (ETS) strategy*: Node x first eliminates all quorums that have at least one node in DQL_x. One of the remaining quorums is randomly selected to perform *query/update* operations. If at least one server sends an ACK in the case of an update, or a $NON\text{-}NULL$ value in the case of a query, the operation is said to be succeed. If all quorums get eliminated before there is a success, the *query/update* operation is said to be failed.

3. *Hybrid strategy*: It uses ETS for updates and STE for queries.

The STE strategy tries to maximize availability of information at the expense of accuracy whereas the ETS strategy tries to maximize accuracy of information at the expense of availability. The hybrid strategy does not try to recover from a timeout access, and fails to return the most recent information in all cases.

Probabilistic Quorum Systems (PAN)

Luo et al. [96] have proposed a protocol, called *Probabilistic quorum systems for Ad hoc networks (PAN)*. The protocol uses a gossip-based multi-cast protocol to disseminate updates in a probabilistic quorum system. A probabilistic quorum system [99] relaxes the intersection property of the strict quorum system, such that write and read quorums intersect only with high probability.

The authors assume a subset of network nodes termed Storage Set (STS) are elected to hold shared data in a replication fashion. Any node $i \in STS$ is called server, whereas the rest of the nodes are called clients. The set STS may change from time to time due to frequent network disconnections, joining, and leaving of nodes. It is not necessary that either the client or the server know all the members of the STS.

The probabilistic system includes two sides: a client protocol and a server protocol. A client node sends a query or an update to an arbitrary server in the STS called *agent* for that client. The *agent* performs a corresponding operation of the server protocol.

The server protocol maintains two types of quorum: a *write quorum* accessed by an update, and a *read quorum* accessed by a query. The server protocol has two entities, which are:

1. *Server update*: The agent propagates the update information within the STS with the aid of other servers. Each receiver receiving the update message stores the updates temporarily in the buffer. The Update process is executed every t seconds to disseminate the message stored in the buffer. Any server that receives this update message using a gossip protocol forwards it to randomly chosen receivers based on the current membership of the STS.

2. *Server query*: After receiving a query message from the client, the agent sends it to other servers. Upon receiving the message, each server belonging to the

read quorum, responds with its own copy of the data item if its version is more recent than the one of the agent.

The gossip mechanism employed to spread update operations prolongs the update propagation time and results in high update loss and update conflict problems in dynamic ad-hoc networks.

Virtual backbone

Liang and Haas [92] have proposed a virtual service backbone, which comprises server nodes that serve only as containers for location storage and retrieval [98, 111]. The server nodes are members of quorum groups. When a node moves, it sends an update to one quorum containing the nearest backbone node. A source node then queries the quorum containing its nearest backbone for the location of the target node.

The authors have adopted a multi-level ad-hoc routing scheme, in which packets are sent from the source node to the destination node through a set of backbone nodes. The servers are dynamically created and terminated as the network topology changes to ensure network-wide service availability.

The authors have defined *r-zone* for each node i as follows: $r{-}zone = \{j | d(i,j) \leq r\}$, such that r denotes the guaranteed maximum hop count for a node to its nearest database, and $d(i,j)$ is the distance in terms of number of hops between two nodes i and j. They have tried to find a set of database servers with minimum cardinality such that every node in the network is within at least one database server's *r-zone*. Each node monitors the identity and connectivity of other nodes within its *r-zone* and calculates *dependency_number*, which denotes the number of *panic nodes* that are within r hops from the node. A *panic node* is the node, which there is no database server within its *r-zone*. In contrast, it is called *normal node*. If a node i has high *dependency_number*, it creates the service and joins the virtual backbone. If a new server appears in its *r-zone* and there is no *panic node* in the *r-zone*, node i will become a *normal node*. Database servers are deleted in the region where there are too many of them. This can be achieved by merging two databases that are within a threshold distance of *D hops*.

In this strategy, the quorum system depends on a topology-based routing protocol for the virtual backbone, which tremendously increases the implementation complexity. In addition, ad-hoc networks are subject to frequent network partitioning, link

and node failures. In such environments, quorum systems may suffer from low availability. Update and query operations may be performed at non-intersecting quorums, which may disable the quorum system.

3.1.3 Access frequency-based data replication protocols

SAF, DAFN, and DCG methods

Hara [57] has proposed three allocation methods to improve data availability: *Static Access Frequency (SAF)*, *Dynamic Access Frequency and Neighborhood (DAFN)*, and *Dynamic Connectivity Based Grouping (DCG)*. These methods make the following assumptions: (1) each node has finite memory space to store replicas, and (2) the access frequency from each node to each data item is known, and does not change. Each node maintains replicas of data that is frequently requested. Replicas are relocated during a certain period, called the relocation period. Replica allocation is determined based on the access frequency from each node to each data item.

Table 3.1: Access frequencies to data items

Data	M_1	M_2	M_3	M_4	M_5	M_6
D_1	0.65	0.25	0.17	0.22	0.31	0.24
D_2	0.44	0.62	0.41	0.40	0.42	0.46
D_3	0.35	0.44	0.50	0.25	0.45	0.37
D_4	0.31	0.15	0.10	0.60	0.09	0.10
D_5	0.51	0.41	0.43	0.38	0.71	0.20
D_6	0.08	0.07	0.05	0.15	0.20	0.62
D_7	0.38	0.32	0.37	0.33	0.40	0.32
D_8	0.22	0.33	0.21	0.23	0.24	0.17
D_9	0.18	0.16	0.19	0.17	0.24	0.21
D_{10}	0.09	0.08	0.06	0.11	0.12	0.09

In the SAF method, each node arranges data items in the descending order of their access frequencies. The access frequency of each mobile node to each data item is shown in Table 3.1. The nodes do not need to exchange information with each other for replica allocation. However, nodes with the same access characteristics may allocate the same replicas, which results in low data availability. For example, in figure 3.1, D_7, D_8, D_9, D_{10} are not allocated.

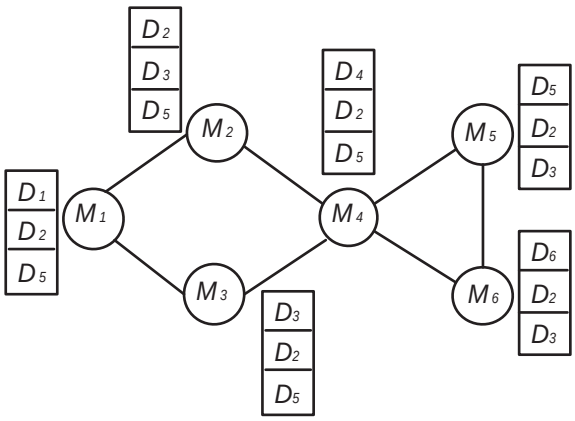


Figure 3.1: An example of executing the SAF method

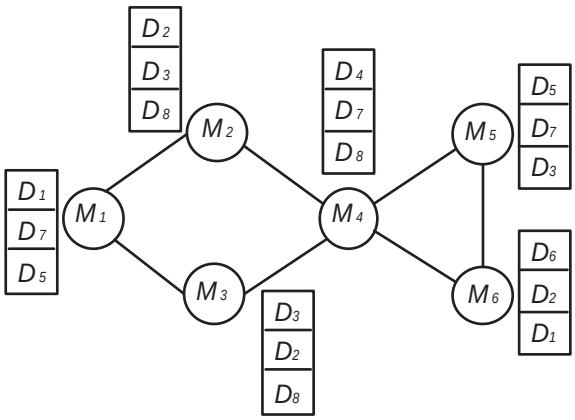


Figure 3.2: An example of executing the DAFN method

In the DAFN method, each node determines the allocation of replicas using SAF. It then eliminates replica duplication between all neighboring nodes i and j. The following procedure is repeated in the order of breadth first search, starting from the node with the least identifier:

- If i holds an original data item and j holds a replica of that data item, j will replace this data item with another data item that has the next highest access frequency from j.
- If both and j hold replicas of the same data item, the node which has a lower access frequency of that data item replaces it with another data item that has the next highest access frequency from it.

The DAFN method increases data availability compared to the SAF method. However, it comes with the cost of higher overhead, because at each relocation period, nodes exchange information and relocate replicas. Moreover, the author has shown that DAFN does not completely eliminate replica duplication among neighboring. Figure 3.2 shows an example of executing the DAFN method in the environment given by figure 3.1. We can see the replica duplication of D_8 between M_2 and M_4 and between M_3 and M_4, and D_7 between M_4 and M_5.

Table 3.2: Access frequencies of groups

Data	G_1	G_2
D_1	1.29	0.55
D_2	1.87	0.88
D_3	1.54	0.82
D_4	1.16	0.19
D_5	1.73	0.91
D_6	0.35	0.82
D_7	1.40	0.72
D_8	0.99	0.41
D_9	0.70	0.45
D_{10}	0.34	0.21

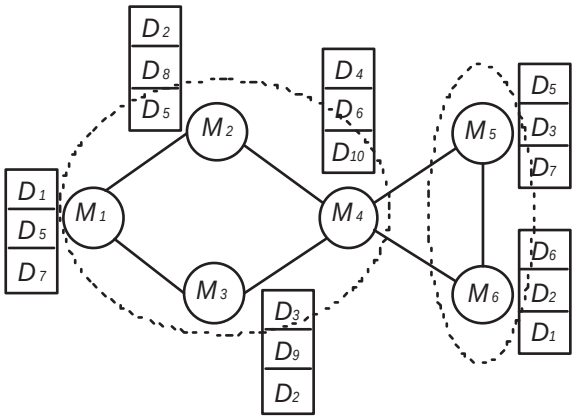


Figure 3.3: An example of executing the DCG method

The DCG method shares replicas in larger groups of mobile nodes than the DAFN method that shares replicas among neighboring nodes. It creates groups of mobile

nodes that are bi-connected components [1] in the network. A given group is bi-connected component if the removing of a single node or link will not split up that component into disjoint subgraphs. These groups are considered to have high stability in terms of network connections. In each group, an access frequency of the group to each data item is calculated as a summation of access frequencies of nodes in the group to the data item. In the order of the access frequencies of the group, each data item is allocated at a node from which the access frequency of the data item is the highest compared with other nodes in its group that have free memory space. After allocating replicas of all the data items, if there is still free memory space at some nodes in the group, replicas are allocated in the descending order of access frequencies in those particular nodes until the memory space is full. Figure 3.3 shows an example of executing of the DCG method, in which the mobile nodes are divided into two groups: $G_1 = \{M_1, M_2, M_3, M_4\}$ and $G_1 = \{M_5, M_6\}$. Table 3.2 shows the access frequencies of the two groups, which are calculated from Table 3.1. Compared with the DAFN method, the DCG method shares more replicas and, thus the data availability is expected to be higher than the DAFN method. However, this method consumes more time and overhead, because the construction of bi-connected groups is carried out at every relocation period.

E-SAF, E-DAFN, and E-DCG methods

Hara [58] proposes extensions to the methods presented in [57], to handle periodic and aperiodic updates. The update of data item is done only by the node that holds the original data item. Replicas of a data item become invalid if the nodes that hold them cannot connect to the node holding the original copy, because the update is not propagated.

In the periodic update, each data item is updated at constant interval. The extensions consider data access frequencies and the time remaining until each data item is updated next. They define the PT_j value for each data item D_j. PT_j denotes the average number of access requests which are issued for D_j until D_j is updated next. It is defined as the product of the access frequency of the data item and the time remaining until the next update. The extended methods *E-SAF*, *E-DAFN*, and *E-DCG* work in the same way as *SAF*, *DAFN*, and *DCG* respectively, except they use PT values instead of the access frequencies.

In the aperiodic update, when a node issues a request, the request succeeds on

the spot if the node holds the original data item. If the node issues a request to a data item of which the node does not have the original, the request is processed by the following steps:

1. The requesting node broadcasts a *data search packet* in the network.
2. A node that receives the *data search packet* checks whether it has the original or a replica of the data item requested by the requesting node.
   - If it has the original, it returns a message notifying the requesting node of that fact and whether the update has occurred or not. At the same time, it stops the broadcasting of the *data search packet*.
   - If it does not have the original but has a replica, it returns a message notifying the requesting node and continues the broadcasting of the *data search packet*.
   - If it does not have either the original or a replica, it does not reply to the requesting node, and continues the broadcasting of the *data search packet*.
3. When the requesting node receives a *reply packet*, it performs as follows:
   - If the *reply packet* is from the owner of the original, and the requesting node has a replica of the target data, and the update has not occurred, then it accesses the replica. Otherwise, it sends a *data request packet* to the owner of the original.
   - If the reply packet is not from the owner of the original but from the owner(s) of replica(s), the following two cases can occur:
     (a) If the requesting node has a replica, it makes an interim access to its own replica (interim access becomes a dirty read when it finds that an update has occurred)
     (b) Otherwise, the requesting node sends a *data request packet* to one of the owners of the replicas, e.g., the nearest owner to the host.
   - If no *reply packet* is received, the following two cases can occur:
     (a) If the requesting node has a replica, it makes an interim access to its own replica.
     (b) Otherwise, the request fails.

The extended methods have the same drawbacks as their original ones. Moreover, the data items are not sent to replica servers when they are updated but the update operations are performed locally, which increases the number of requests that access invalid data items.

AL and GM methods

In [59], Hara has extended the *SAF* and the *DCG* methods and proposed the *AL (Access Log)* and the *GM (Group Management)* methods respectively.

In the AL method, each mobile node holds an access log table, which contains the pairs of a data identifier and a list of nodes that hold the data item corresponding to the data identifier. When a node sends an access request to data item D_j, the request succeeds by accessing the original or a replica of the target data item held by a mobile node M_i, the requesting node inserts M_i at the top of the list of node identifiers corresponding to D_j. If M_i already exists in the list, the old one is deleted from the list. The size of each list is limited to L items and the last one is deleted if the size exceeds L. When a node issues a request to D_j, it successively unicasts the request to the mobile node according to the order in the list for D_j in its own AL table until the request succeeds. If all the requests fail, it broadcasts the access request in the network.

In the GM method, each mobile node holds a location management (LM) table, which consists of pairs of a data identifier and a list of nodes that hold the data item corresponding to the data identifier. It also holds a gateway (GW) information that represents gateway nodes in the group to which the node belongs. Here, a gateway node represents a node that connects to at least one mobile node belonging to other groups. A mobile node sends its request for data item D_j by executing the AL method. If all the requests based on the LM table fail, the requesting node successively unicasts the request to a gateway node according to its own GW information until the request succeeds. Figure 3.4 shows a case when mobile node M_1 issues an access request to D_2. As there is no node that holds D_2 in the same group, M_1 unicasts the request to gateway node M_4. The gateway node forwards the received request to the neighboring nodes in other groups M_5. M_5 forwards the request to M_6 according to its own LM table. If the requests fail, the node unicasts the request to gateway nodes in its group. This process repeats until the request succeeds. If all the above processes result in failure, the requesting node broadcasts the access request over the entire network.

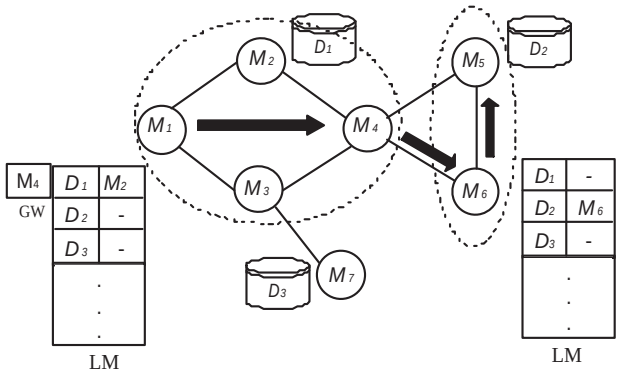


Figure 3.4: An example of executing the GM method

AL+ and GM+ methods

Hara [60] has extended the methods in [59] an proposed *AL+* and *GM+* to adapt to an environment where each data item is updated. He has assumed that every node knows the node that holds the original of each data item. Thus, a requesting node first unicasts the request to the node holding the original. If this attempt fails, the extended methods try to find nodes that hold replicas of the target item. In the extended methods, the list for each data item in an LM table is managed to give higher priorities to nodes that hold replicas with larger time stamps. In an environment with data update, the replica that is first found according to the LM table may not be the latest version among replicas held by the connected mobile nodes. So, the extended methods execute one of the following two actions: First, if a replica is found, the data access process finishes. Second, if the first action does not succeed, the requesting node continues the data access process until it finds k replicas. After finding k replicas, it tentatively accesses the replica of the latest version among them.

The readers can easily conclude that the AL, GM methods, and their extensions (AL+, GM+) are costly in terms of storage space and message overhead, which make them unsuitable for ad-hoc networks.

Greedy-S, OTOO, and RN methods

Yin et al. [148] have stated that although being able to achieve low query delay by replicating most data locally, the data accessibility is reduced if many nodes replicate the same data items, while some data items are not replicated by any node. On

the other hand, if the nodes do not replicate the same data that neighbor nodes have already replicated, the data accessibility is increased. However, the query delay in this case is increased since some nodes may not be able to replicate the most frequently accessed data, and have to access it from neighbors. To balance the tradeoffs between data accessibility and query delay, the authors have proposed three methods to achieve this aim. The methods assume that each node only has a memory size of C memory space to host data replicas. The access frequency of node N_i to a data item D_j is denoted by a_{ij}, s_i is the size of D_i, and f_{ij} is the link failure probability between node N_i and N_j.

In the greedy method, as smaller data items require less memory size, thus replicating them can save the memory size for other data items. The authors propose the following function $AF_i(k) = (\frac{a_{ik}}{s_k})$. Each node allocates data items in descending order of AF_i until no more data can be replicated to the memory. This method replicates more frequently accessed data locally. However, it does not consider the cooperation between neighboring nodes.

In the One-To-One Optimization (OTOO) method, each mobile node only cooperates with at most one neighbor to decide which data to host. Each node N_i calculates the Combined Access Frequency (CAF) value of N_i and N_j to data item D_k at N_i, denoted as $CAF1_{ij}(k)$.

$$CAF1_{ij}(k) = \frac{(a_{ik} + a_{jk} \times (1 - f_{ij}))}{s_i}$$

Then, it allocates the data items in descending order of *CAF1* until no more data items can be replicated. Although, this method improves the data accessibility, but it may happen that node N_i should host D_j but the node that hosts D_j is not reachable to N_i because of network partitions.

In the Reliable Neighbor (RN) method, for a mobile node, if its communication links to other nodes are stable, more cooperation with these nodes can improve the data accessibility. In other words, each mobile node contributes part of its memory to hold data items for other reliable nodes, i.e. the link failure probability is less than a given threshold value. Let nb_i be the set of the N_i's reliable neighbors. The total contributed memory size of N_i, denoted as $Cc(i)$, is set to be

$$Cc(i) = C \times \min(1, \sum_{N_j \in nb(i)} \frac{1 - f_{ij}}{\alpha})$$

where α is a system factor. The most interested data are first allocated to the up $(C - Cc(i))$ memory space. Then all the rest of the data are sorted according to $CAF2$ values. The $CAF2_i(k)$ function of N_i to D_k is defined as:

$$CAF2_i(k) = \frac{(\sum_{N_j \in nb(i)} a_{jk} \times (1 - f_{ij}))}{s_k}$$

The proposed methods try to increase data accessibility by considering only neighboring nodes. The authors consider only query requests that traverse at most one hop. They do not consider larger stable group, in which more data items can be shared, and high data accessibility can be obtained. In such groups, the query delay can be limited to a constant number of hops that does not affect the query latency allowed by the system.

3.2 An Energy-aware data replication protocol: Expanding Ring Replication (ERR)

The authors [136] have proposed a replication scheme, called *Expanding Ring Replication (ERR)* that combines the push-based and the pull-based data delivery approaches. In the pull-based approach, when a node wants to access some data items, it broadcasts to its neighbors *interest advertisements* message, containing a description of the data items required. If the request is not satisfied within a given time period, the node initiates an information request to the server. In the push-based approach, the data server measures the frequency of requests f_i for each data item D_i. If f_i exceeds a threshold value, λ_i, set by the server for D_i, the server decides to replicate the data on one or more capable nodes in the network. The capability function considers parameters such as available memory space, remaining battery power, and processing power. The data server maintains a set of hop count values, S_i, for each data item, D_i. These hop count values represent the number of hops from the server where the data should be replicated.

To replicate the data in the network, the server probes the nodes $h \in S$ hops into the network, soliciting their capabilities to replicate data items. When the probe packet reaches a node $h \in S$ hops away, the node computes its capability to replicate

a data item. If it is capable, it replies back to the server, sending a probe acknowledgement packet. The probe acknowledgement packet contains the list of items the node is interested in or has the memory space to cache. The server then directly sends the data items to the nodes from which it receives a probe acknowledgement.

In this solution, the authors do not consider data update or how a client node should behave if the data server is unreachable.

3.3 Partition-aware replication protocols

Network partitioning decreases data and service availability to a large extent. A good replication strategy should be designed to maintain data and service availability at a desired level even during frequent disconnections and network partitioning.

3.3.1 Service Coverage

Wang et al. [142] have proposed a set of run time algorithms to ensure the availability of centralized services to all mobile nodes. The authors assume that nodes move according to correlated mobility patterns, called the *Reference Point Group Mobility (RPGM)* [143]. In this model, each group has a logical group center called the reference point, which defines the movement of all the nodes in the group. They have extended the RPGM model and proposed a *Reference Velocity Group Mobility (RVGM)*. In RVGM, each group has its own mean group velocity, and all the nodes within a group have a velocity that slightly deviates from their mean group velocity.

The authors also assume that each node is aware of its position and its velocity via GPS or through received signal power measurements. In order to guarantee the service availability, server nodes need to predict the occurrence of network partitioning to replicate the services or data that they hold in advance. Server nodes know the client velocities as these are piggy-backed on the client requests. They use a *sequential clustering algorithm* to identify the different mobility groups, which are used to predict the time and location of network partitioning. By calculating the time of service replication, a server can replicate the service onto the partitioned nodes beforehand. In order to minimize the number of service instances deployed in the network, servers also run a *distributed grouping algorithm* at regular *service discovery intervals* to discover a set of stable servers (servers with a stable connectivity that is unlikely

to be disconnected). By doing so, the servers in the same stable group monitor each other's presence. As an arbitration, the server with the highest *id* continues its service, and the others automatically terminate the service instances.

At each *service discovery interval*, a *distributed grouping algorithm* is run by clients to discover mobility group membership. After the run of the algorithm, a client constructs its stable group and discovers a set of servers. The client selects the best server among the discovered servers; the best server is the one whose relative velocity will allow it to stay in the client group for the longest time.

The proposed algorithms guarantee the replication of the services before the occurrence of network partitioning, which ensures that the service is available in each network partition with the cost of one server per mobility group. However, the proposed solution has the following drawbacks. First, client nodes need frequent accesses to at least one of the servers in order that the server nodes run the *sequential clustering algorithm*, which incurs a heavy load on these clients characterized by their limited resources. Second, the solution addresses only for group mobility model. Third, the solution depends on expensive and bulky positioning system (GPS).

3.3.2 Hauspie's Protocol

Hauspie et al. [65, 66] have proposed a new metric to detect network partitioning without using GPS. The metric is based on finding a set of disjoint paths between a client node and a server node. A set of disjoint paths is a set of paths that have no common nodes except the client node and the server node [33, 88, 90, 95]. The decision to replicate a service or a data item is taken when the connection between a client and a server becomes bad in terms of reliability, bandwidth, delay,$\cdots$, etc. Replicating the service or the data item on a node that is closer to the client node can enhance the quality of connection between the client and the server nodes.

Given w is a server node, each client node v periodically calculates the reliability metric $R_k(v, w)$. The metric first determines the set of k$-$sub$-$optimal paths between v and w, $SOP_k(v, w)$ providing that $k > 0$. A path p is k$-$sub$-$optimal if and only if

- p is loop free.
- $|p| < d(v, w) + k$, where $d(v, w)$ is the distance of the optimal path between v and w.

Then, it determines the set of parts of $SOP_k(v, w)$ containing only disjoint paths. Such a subset is denoted by $DSP_k(v, w)$. The robustness of the link is given in equation 3.1.

$$R_k(v, w) = \max_{C \in DSP_k(v,w)} \left\{ \sum_{p \in C} \frac{1}{|p|} \right\} \tag{3.1}$$

Based on this metric, each client node v executes the following algorithm.

1. v sends a *link evaluation request* to w using an efficient routing protocol.
2. w replies by a *link evaluation reply* using a flooding algorithm like [88, 137] with a TTL of $d(v, w) + k$.
3. v checks if the received path is disjoined of all the stored paths. If so, it stores it. Otherwise the path is discarded.
4. When a given time out is elapsed, v evaluates $R_k(v, w)$.

When $R_k(v, w)$ falls below a given threshold during a determined amount of time, the replication process is triggered. The replication is not only for predicting network partitioning, but also when the connection becomes bad. Although the algorithm does not use GPS, but the computation of $R_k(v, w)$ metric requires a lot of network load, memory and CPU. Moreover, a client node stores many disjoint paths to the same server, which represents a heavy load on it.

3.3.3 Chen's Protocol

Chen et al. [26, 25] have proposed a data lookup service, in which data availability information is exchanged between nodes of a connected group. Each node periodically broadcasts an advertising message called *ad* to each member of the group. The *ad* message contains a sequence number, node's capability (free space, remaining power, and processor utilization), and the available data items at the node. Each node, upon receiving the *ad* message, updates its local *Data lookup table*. When duplicated data is found at the receiver node and if its address is lower than the advertising node's address, the local copy of the data is deleted.

Each node of the group uses location information provided by GPS, and those received by other nodes of the group to calculate the movement of each member in

the group, and hence to predict group partitioning and replicate data items of the future separate partitions on other nodes that are still connected to the node.

Since this strategy predicts the occurrence of network partitioning ahead of time and considers the power constraints of mobile hosts. However, this strategy suffers from the following disadvantages. First, GPS is not suitable for small devices due to its high power consumption. Second, the network is highly loaded due to continuous information exchange. Third, data access delay is not bounded. Fourth, the authors have not given a definition for the group of nodes. One can assume that the whole connected network is also a group.

3.3.4 DAFN-S1, DAFN-S2, and DCG-S1 methods

Hara [61] has proposed three methods, DAFN-S1, DAFN-S2 and DCG-S1, to decrease the effect of network partitioning. The methods aim to eliminate duplicate replicas of data items between two nodes if the wireless link that connects them is stable. The methods assume that each node through GPS knows the speed and the direction of the movement. The wireless link is disconnected if the distance between the neighboring nodes becomes longer than the communication range. The time t_{ij} at which two nodes i and j will be disconnected can be estimated by their movements. The stability of the wireless link between the two nodes i and j is then given by:

$$B_{ij} = \begin{cases} 1 & t_{ij} > T \\ \frac{t_{ij}}{T} & otherwise \end{cases}$$

where T is the relocation period.

DAFN-S1 (DAFN - Stability of radio links: 1)

DAFN-S1 differs from DAFN [57] in the fact that stability of wireless links are taken into account when eliminating duplicate replicas between neighbor nodes. For each neighboring nodes i and j, if there is a duplication of a data item between them and B_{ij} is more than a threshold value, one of the redundant data item is eliminated. When B_{ij} is smaller than the threshold value, the link (i, j) is simply ignored.

DAFN-S2 (DAFN - Stability of radio links: 2)

DAFN-S2 is a variation of DAFN-S1. It uses another evaluation method to eliminate duplicate replicas. First, the node whose replica is to be replaced is chosen with the same method as described in DAFN-S1. Let the access frequency to this data item at this node be p. This node calculates the value of $((1 - B_{ij}) \times p)$, which represents the probability that this node will access the data item after being disconnected from the other node. If this value is more than a threshold value, the node replaces the data item.

DCG-S1 (DCG - Stability of radio links: 1)

DCG-S1 works in the same way as DCG except that it takes into account the stability of wireless links when grouping the nodes together. Two nodes i and j are considered to be connected (i.e., the link (i, j) is stable) if B_{ij} is greater than a threshold value. The set of bi-connected groups are created by considering only stable links.

The three methods DAFN-S1, DAFN-S2, and DCG-S1 considerably alleviate the issue of network partitioning but they are not partition-aware strategies. The decision to replicate is taken only at every relocation period, and the author has not explain how the methods deal with frequent topology changes, and especially when a network partitioning occurs between two successive relocation instants. For example, when the network partitions, queries that arrive between the occurrence of network partitioning and the next immediate relocation period, will fail. Moreover, DAFN-S1, DAFN-S2, and DCG-S1 inherit the same advantages and disadvantages of DAFN and DCG respectively.

3.3.5 DRAM protocol

Huang et al. [71] have proposed DRAM, a replication protocol to alleviate the reduction of data accessibility. The protocol exploits the group mobility of nodes for replication allocation. It uses the RPGM [143] model to identify network partitions. As network connectivity changes frequently, DRAM is executed at every relocation period. DRM consists of two phases: (1) the *allocation unit construction phase*, and (2) the *replica allocation phase*.

In the allocation unit construction phase, each node exchanges its motion information with nodes located within a determined hop count. Then, a clustering algorithm clusters mobile nodes with similar motion behavior into mobility groups. Clusters that are likely to be connected are merged into an allocation unit. An allocation unit is a set of mobile nodes which share their storage and do not store repeated data items unless all data items have been allocated in this allocation unit. Each node in its broadcast zone that has the lowest node identifier is chosen as the master of the broadcast zone. Zone masters then cluster member nodes having the same motion behaviors into clusters. Each member then sends status messages to the cluster master and, when it is unable to reach the master, enters the initial state and joins another group. The master also removes nodes that do not send status messages from the cluster. The merging of clusters that are about to be connected into a big allocation unit makes the mobile nodes within the resulting cluster share more data items and hence the data accessibility is improved.

In the replica allocation phase, the replicas of all data items are allocated in accordance with the access frequencies of the data items and the derived allocation units. The allocation weight of data item D_j in allocation unit C_x in timestamp $T(k)$ (denoted as $w_j^x(k)$), is the expected number of data accesses from all mobile nodes in C_x before the next update of D_j. $w_j^x(k)$ can be obtained as follows:

$$w_j^x(k) = f_j^x \times (U_j - k),$$

where

$$f_j^x = \sum_{\forall M_i \in C_x} f_{ij}$$

where U_j the timestamp of the next update of D_j. Since the update of each data item is periodic, U_j can be predicted in advance. All data items are allocated in C_x according to their allocation weights in C_x in descendent order.

The proposed protocol works only for group mobility. If random motion of nodes occurs, the group joining and leaving operation may become an overhead in the system. Moreover, if there is a large number of nodes that have diverse movement patterns, the groups formed may be small in size.

3.3.6 Jorgic's protocol

Jorgic et al. [77] have proposed localized algorithms to detect critical nodes and links that could partition the network. A node makes a decision to determine critical nodes or links based on limited local knowledge, called the *k-hop knowledge*. The authors define the *k-hop neighbors* of a node A as the set of nodes S such that: the shortest route between each node in S and A has k or less hops. Nodes collect k-hop knowledge by sending *hello* messages to its neighbors containing the graph of their (k-1)-hop neighbors. Nodes may add geographic position information in the *hello* messages.

A node A is *k-critical* node if the subgraph of k-hop neighbors of A (assuming that A does not exist) is disconnected. Based on information used (i.e. topologically or positionally), the corresponding algorithms, which are used to detect k-critical nodes, are referred to as being *k-top_critical_node* and *k-pos_critical_node* algorithms. If a node is globally critical, localized algorithm will detect it as such.

If topological information is only used, the algorithm, referred to as *k-link_top_critical* algorithm, defines a link AB as a critical if the sets of k-hop neighbors of A and B (assuming that the link AB does not exist) are disjoint. Otherwise, if position information is used, the corresponding *k-link_pos_critical* algorithm is defined as follows. AB is a critical link if the sets of k-hop neighbors of A and B (assuming that the link AB does not exist) are disjoint, and there are no two nodes, one from each set, which are neighbors.

The localized algorithms monitor the nodes and the links of a given route. If such a node or a link is detected, an alternative service is searched or the service is replicated. However, the proposed algorithms may detect some nodes or links as critical although they may not be globally critical. The accuracy of the algorithms can be increased if k is increased, but this comes with the cost of additional overhead. Moreover, these algorithms are executed even if the network is in a stable state, which represents a vain effort.

3.4 Scalable data replication protocols: Cluster-based approach

3.4.1 Distributed Hash Table Replication (DHTR)

Yu et al. [150], have proposed an optimistic replication management system, called Distributed Hash Table Replication (DHTR). A replication system is said to be optimistic if all replicas are allowed to be independently written and read during update propagation.

DHTR is built on a cluster-based architecture, in which all mobile nodes are put into multiple non-overlapping groups and a cluster head is selected for each group to handle the control packets for replica query and update propagation. The DHTR system is mainly composed of two elements: replica managers and cluster heads. Replica managers are mobile nodes that hold replicas. They accept query requests from other mobile nodes and reply with the requested files. A replica manager periodically communicates with its cluster head to register and update replica information. The structure of DHTR is illustrated in Figure 3.5.

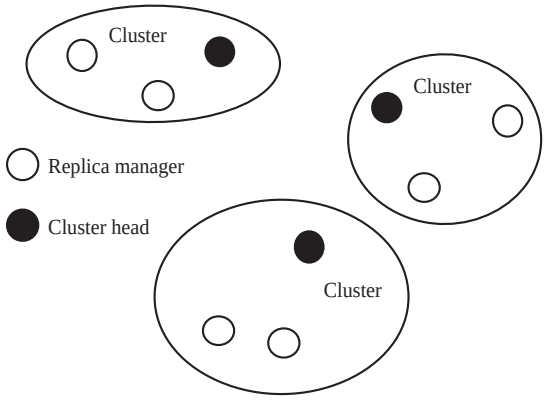


Figure 3.5: Structure of DHTR

The clusters are formed by executing an election algorithm similar to the highest-connectivity leader election algorithm. The authors assume that the number of clusters is set to M and that all the nodes are aware of M and the total number of mobile nodes. All nodes flood their capacity, including current residual energy, and free storage space, and their node IDs. After receiving all the broadcast information, each mobile node puts the node IDs into a sequential list sorted by their capacity, with

node ID used to break ties in capacity. The M nodes at the top of the list become the initial cluster heads and their specific location in the sequential list becomes their cluster ID. The number of clusters remains constant during execution to maintain the stability of the replica information directory.

After the initial phase, each cluster head forms its cluster domain by enrolling replica managers and other clients. Cluster heads synchronously flood the whole network with its node ID and cluster ID in a $CH\_ANNOUNCE$ message. The mobile node joins the cluster head whose $CH\_ANNOUNCE$ message arrives earliest. When a cluster head receives a $CH\_ANNOUNCE$ message, it extracts related information and adds an entry in its local *Cluster-Node ID Mapping (CIM)* table. The function of the *CIM* table is to map Cluster IDs and corresponding cluster head node IDs.

After the creation of the clusters, each replica manager transfers information about the replicas it holds to its cluster head, which puts it into a *Local Replica Cache (LRC)*. A Replica Keeper for an object O is a cluster head that stores all the replica location information for O in a *Global Replica Cache (GRC)*. The role of the GRC is to record the distribution of particular object replicas in various clusters.

The movement or failure of a replica manager triggers the modification of the LRCs and GRCs. So, a replica manager must send hello packets at fixed intervals to its cluster head in order to keep the cluster head's LRC up-to-date. When a manager decides to associate with a new cluster head at a shorter distance, it issues a register request to the new cluster head and an un-register message to the former one. If a cluster head receives an un-register message or fails to collect the hello message during three consecutive intervals, it assumes the manager has left the cluster or disconnected from the network and flushes all related replica information from the LRC. Furthermore, if no further replicas of that object remain in the cluster then the cluster head has to send an un-register message to the object's replica keeper to update its GRC.

For a query request, a client sends the request to its cluster head. When the home cluster head receives the request it first searches its LRC for a local replica manager with the required object. If no LRC is found, the query is forwarded to the replica keeper of the required object. The replica keeper refers to its GRC and sends the query request to the cluster head. The request finally reaches the replica manager via that cluster head. Updates to an object are first sent to the replica keeper for that object and then disseminated to other replicas after it is accepted by the keeper.

This solution has some disadvantages. First, the replica keeper is not well explained. One can assume that the replica keeper is a centralized node, which makes the system less resilient to node failure and network partitioning. Second, the authors assume that the number of clusters does not change.

3.4.2 A Clustering-Based Data Replication Algorithm (CDRA)

In [154], a set of mobile nodes are grouped into clusters. The clusters should guarantee the following properties: they are composed of stable links, and there is at least one stable path between any pair of nodes in a cluster. A link is said to be α-stable if the connectivity probability between node i and its neighbor j is greater than α. A path is said to be α-stable if the product of the connectivity probability of the links that compose the path in question is also greater than α.

Every cluster head maintains states of all other cluster heads in the networks. When a node requests to access a data item, the node broadcasts the access request in the whole of cluster C that the node belongs to. If there are some replicas of the data item in the cluster, the closest replica node serves the access request. If there is not replica node for the requested data object in C, the request is propagated from the cluster head of C to all other cluster heads. If there is replica in some cluster C', the cluster head of C' sends the data to the cluster head of C, and the cluster head of C sends the data to the requesting nodes. The node which has requested the data item is chosen to be a replica for this data item.

The write requests for the data object are propagated to all cluster heads whose cluster has the replica of the data object, and then are forwarded to the replica nodes in the cluster. If the write request is granted, data update message forwards to all the replica nodes in the same way.

The proposed protocol suffers from the following drawbacks: First, the maintenance of clusters is costly since an α-stable path must exist between any two nodes of the same cluster. Second, the replicas of a given data item are created only in clusters that contain requesting node of that data item. Third, whenever a data item is updated, the new version of the data item is propagated to all cluster heads of the network. If data items are updated at high rate, the throughput for data update may reduce due to channel access contention. Fourth, the update operation is performed in two phases, which incurs high update delay and update cost.

3.5 Scalable data replication protocols: Location-based approach

Nodes in ad hoc networks operate on low-power batteries. A replica server may respond to the requests of many clients, which leads that it may quickly drain out its battery. Moreover, as load increases the server is likely to become a bottleneck. In some cases, the subset of nodes that play the role of replica server is fixed, such static membership may result in low data availability because the replica servers may drain out of power very quickly.

To select a new replica server, some of the replication protocols have to scan the entire network to find out the appropriate node. Moreover, they use a topology-based routing protocol to deliver the query and the update packets to their destinations, which incur high message overhead and increase the implementation complexity.

To deal with these issues, many solutions suggest that a node does not need to know the ID of its replica server to whom it should send its update and query packets, but it only has to know to where should a node send the packets. Using a geographic routing protocol, update and query packets can be sent toward fixed regions or positions and not toward nodes. Some or all nodes inside the regions or close to the positions can act as servers. Using this method, the dynamic nature of the ad hoc network is masked by a static scheme.

3.5.1 Geography-based Content Location Protocol (GCLP)

Tchakarov and Vaidya have proposed the *Geography-based Content Location Protocol (GCLP)* [135]. GCLP uses similar approaches described in [9, 105]. [105] proposes propagating the advertisements and queries in cross-shaped trajectories, thus guaranteeing two intersections. Queries are answered by nodes, called content location servers (CLS), at the intersection of the advertising and query trajectories.

Content advertisement is performed by periodically sending update messages through the network in four geographical directions (north, south, east, west). If a content location server receives multiple advertisements for a particular resource, it will only forward updates from the content server closest to it. To locate content on the network, a client sends out a query message through the network in a manner identical to the update messages (i.e. sent in the four geographical directions).

This protocol does not guarantee intersection of update and query messages. Attempting to route queries and updates in a straight line can become difficult when there are obstacles that create holes in the network topology. Second, it incurs high overhead since updates are sent periodically, and each message is sent in the four geographic direction.

3.5.2 Geographic Hash Table for Data-Centric Storage (GHT)

In [120], the Geographic Hash Table system for Data-Centric Storage on sensor networks have been proposed. In this system, a data object is associated with a key. It hashes keys into geographic coordinates, and stores a key-value pair at the sensor node geographically nearest the hash of its key. GHT requires nodes to know their exact geographic location and uses the GPSR [81] routing protocol to identify and to reach a packet's home node (the node closest to the geographic destination). The update and a query operations on the same key k, both hash k to the same location.

This protocol is not well-suited to highly dynamic networks, since it depends on exact location information that become stale very quickly. Moreover, as nodes change positions, the node chosen by the update operation and that by the query operation may be not the same.

3.5.3 Rendezvous Regions (RRs)

In [125], Rendezvous Regions (RRs) incorporated a similar approach to GHT [120], but uses rendezvous regions instead of rendezvous points. The network topology is divided into geographical regions, where each region is responsible for a set of keys representing the services or data of interest. Each key is mapped to a region based on a hash-table-like mapping scheme. A few elected nodes inside each region are responsible for maintaining the mapped information. The service or data provider stores the information in the corresponding region and the seekers retrieve it from there.

The mapping is known to all nodes and is used during the update and query operations. A node wishing to update or query a key obtains the home region responsible for that key through the mapping, then uses a geographic-aided routing to send a message to the region. Inside the region, a simple local election mechanism dynamically promotes nodes to be servers responsible for maintaining the mapped

information.

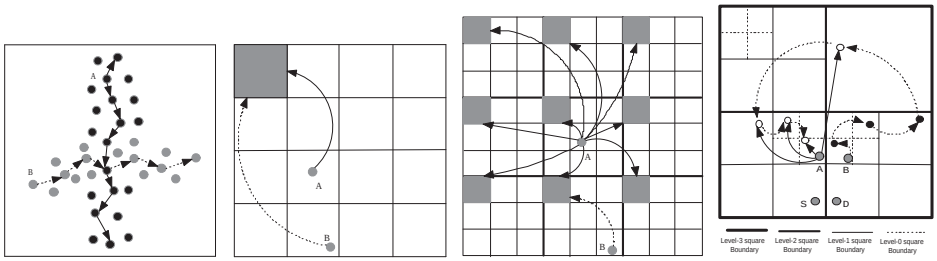
By using regions instead of points, this protocol requires only approximate location information and accordingly is more robust to errors and imprecision in location measurement and estimation than schemes depending on exact location information. A major disadvantage of this design is the single fixed home region. Nodes are not limited in their movements. As a result, nodes may be far away from their home region and their updates may have to travel long distances. Furthermore, even queries from nodes close to the target node must be forwarded all the way to the home region. This can lead to high network load and latency.

3.5.4 Special case: Location services

Location services are special cases of replication protocols when the data items are location information, and a node sends an update packet to server nodes when only it changes its position.

According to Mauve classification [102], the location services are classified according to how many nodes host the service (i.e., *some* or *all* nodes). Furthermore, each location server may maintain the position of *some* specific nodes or *all* nodes of the network. This yields four possible combinations are as follows: *some-for-some*, *some-for-all*, *all-for-some*, and *all-for-all*. In the all-for-all location services, all the nodes flood their location information. Clearly this approach is not scalable. Some-for-some and some-for-all schemes also have problems as they put great burden on nodes selected as servers. The majority of location services proposed in the literature such as: SLURP [145], SLALoM [28], GLS [91], DLM [147], and HIGH-GRADE [151] are classified as all-for-some approach, where all the nodes in the network store some information about other nodes.

When designing a location service, the following basic questions need to be considered: (1) How does a node A choose a set of location servers to store its location information? (2) How should A update these location servers as it moves around? and (3) How does another node B discover the appropriate location server(s) of A to retrieve A's location? Figure 3.6 depicts three structures to deploy location servers, in which the solid line represents the location update path and the dashed line represents the location query path.



(a) Quorum (b) Flat (c) Two-level (d) Multi-level

Figure 3.6: Location server organizations

Quorum-based location services

Quorum-based approach for information replication at multiple nodes acting as repositories is the basic principle of several location services [82, 87, 55, 99]. A location update is sent to a subset of nodes called *update quorum*, and a location query is sent to a subset of nodes called *query quorum*. The subsets are designed such that each query quorum for a node intersects an update quorum for any other node, and the probability of a query success is maximized. The quorum-based location service can be configured to operate as all-for-all, all-for-some, or some-for-some approach, depending on how the size of the quorum is chosen.

In the Column/Row Location Service (CRLS) [130] (see Figure 3.6(a)), the location of each node is propagated in the north-south direction, while any location queries are propagated in the east-west direction. When a node decides a location update is needed, it propagates the location update along the north-south direction, i.e., with the goal of reaching all the nodes in the same column in the geographic area. Each node selected as a location server in the north or south direction broadcasts the update to its one hop neighbors. The update contains the identifier of the next location server in the update direction. When a source node initiates a location request for a destination node, the query is propagated along the east-west direction, i.e., along its row of nodes in the geographic area. The query contains the time of the most recent location known to the source. If a node along the row has a cached location for the destination node that is more recent than the time in the query, it sends a reply packet via geographic forwarding back to the source. The intersection of row and column can be guaranteed by adding outer face [20] of the ad hoc network

to both of them.

Flat hashing-based location services

The home location register (HLR) technique, in which a well-known hash function is used to map each node's identifier to a home region, is employed in SLURP [145]. In SLURP, the network area is divided into a flat grid of squares. Node A selects its location servers by applying a hash function to A's ID and obtains the (x, y) coordinate of a point in the entire area. The square containing that point is called the *home square* for node A. All nodes in that square store A's exact location information. Every time node A moves to a different square, it updates its home square with new location information. For any node B wishes to communicate with node A, the same hash function is applied to node A's ID to obtain A's home square. A query packet is then forwarded to A's home square to retrieve A's location information. This is illustrated in Figure 3.6(b). However, SLURP still suffers from the same disadvantages as Rendezvous regions protocol.

Hierarchical location services

To address the problems of the flat-based approach, SLALoM [28] uses a two-level structure. The entire network is first divided into a flat grid of level-1 squares as in SLURP. The network is then partitioned into various level-2 squares with each level-2 square containing many level-1 squares. Node A selects its location servers by hashing to the same point in each of the level-2 squares. Node A thus has a home square in every level-2 square as in Figure 3.6(c). SLALoM defines home squares near A as the nine level-1 home squares closest to A (i.e., the home square in the level-2 square where A is in, plus the eight home squares in the surrounding level-2 squares). It employs a two-level grained location information, i.e., all the home squares of A know which level-2 square A resides in, and all the home squares near A know the exact location of A. As A moves around, only closer servers need to be updated frequently, whereas remote servers require only infrequent updates. To query node A, node B sends query packet to A's home square in the level-2 square B is in. If that home square is the one near A, B can retrieve A's exact location. Otherwise, the servers in that home square know which level-2 square A is in. The two-level structure reduces the location query cost but increases the cost of updating location servers.

The alternative structure is called the multi-level structure. It is employed by GLS [91], DLM [147], and HIGH-GRADE [151]. A major benefit of maintaining a hierarchy is that when the source and the destination nodes are nearby, the query traversal is limited to the lower levels of hierarchy.

In HIGH-GRADE, the entire network area is called a level H square. The level H square is divided into four quadrants, called the level-$(H-1)$ squares, each of which is further divided into four quadrants as well, so and so on forth, until the entire region is divided into 4^H level-0 square. If the size of the geographical area covered by the network is S. The side length of a level-0 square is denoted by $R = \frac{\sqrt{S}}{2^H}$. Figure 3.6(d) illustrates this hierarchy of squares with an example where $H = 3$.

Servers in HIGH-GRADE store multi-grained information, i.e., each level-i server $(0 \leq i \leq H)$ stores the information of "which level-$(i-1)$ square A is in", and only level-0 servers store the exact location of A. To determine the relative location of A's servers in each level-i square, HIGH-GRADE applies $(H+1)$ hash functions on A's ID. The set of hash points are called Location Server Points (LSPs), denoted by $LSP_{A,i}$ for each level-i square, are shown as white circles in Figure 3.6(d). The location servers are a set of nodes that are closest to each $LSP_{A,i}$. As multi-grained information is stored at each level of LSP. Update are made on level-i only when a node traverses level-$(i+1)$.

When a node B wants to find A's location, it obtains sequentially the level-i potential LSP ($pLSP_{B,A,i}$) by applying the same hash functions to A's ID in B's level-i square. These points are shown as black circles in Figure 3.6(d). IF B and A are co-located in the same level-0 square (i.e., $pLSP_{B,A,0} = LSP_{A,0}$), B can retrieve A's exact location information from the server at $LSP_{A,0}$. Otherwise, the nodes at $pLSP_{B,A,0}$ re-forward the query packet to $pLSP_{B,A,1}$. This process continues until $pLSP_{B,A,i} = LSP_{A,i}$ for some i, where the first location server of A is found. Then, the query is re-forwarded to lower levels LSPs sequentially, i.e., $LSP_{A,i-1}, \cdots, LSP_{A,0}$, where the exact location of A is finally retrieved.

In GLS, each mobile node may designate nodes in each sibling region with IDs closest to its own ID to serve as its location servers. When a node B wants to find the location of A, it sends a query packet toward a node whose ID is the least greater than or equal to the A's ID within the order-1 square. The query packet is then re-sent to the node in the next order square in the grid hierarchy, whose ID is the least greater than or equal to the A's ID within the order-2 square. The process continues

Table 3.3: Notations

n	number of nodes
r	constant
Q	quorum size
v	node speed
γ	node degree (the average number of neighbors of one node)
S	storage capacity
M	the maximum number of servers in DHTR

until a node that has the position information available is found. Such a scheme works fairly well in stationary networks; however, when node mobility is considered, it becomes less efficient. First, in order to find an appropriate location server, a node needs to potentially scan the entire region to find out which node has the closest ID. Second, with node mobility, a new node may appear in a specific region and the original location server may move away from this region; in order to keep the closest ID property of location servers, the location service needs to check the entire region periodically and change location servers accordingly.

DLM partitions the entire network like GLS. There are $H + 1$ levels of squares. The location servers are distributed uniformly across the network, one server in every level-k square. If the server is located in the same level-k square as A, the complete location information is stored. When a node B wants to find the location of A, and the partial address policy is used, a query packet is sent toward the location server of its level-k square. If the complete address of A is found, the query is complete. Otherwise, the server of A indicates which level-j square A is in such that $j > k$.

3.6 Discussion and comparison

In this section, we compare the replication protocols described in this chapter using a common framework consisting of (1) the performance metrics presented in Chapter 2 and (2) the following eight features.

- *Partition-aware:* Does the replication protocol predict network partitions before they occur ?
- *Energy-aware:* Does the replication protocol address node power limitations ?
- *Scalability:* Is the replication protocol scalable ?

Table 3.4: Comparison of data replication protocols not addressing ad hoc issues (Part 1)

	Hauspie [64]	DU, DC [67]	Karumanchi [82]	PAN [96]	Virtual Backbone [92]
Architecture	D	D	D	D	C
Read-only	Yes	No	No	No	Yes
Partition-aware	No	No	No	No	No
Energy-aware	No	No	No	No	No
Scalability	No	No	No	No	No
Robustness	No	No	No	No	No
Localized	Yes	Yes	No	No	No
Routing dependency	None	None	T	T	T
Replication cost	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n/\gamma^r)$
Update cost	N/A	$O(n)$	$O(Q(n+\sqrt{n}))$	$O(n)$	N/A
Query cost	0	0	$O(Q(n+\sqrt{n}))$	$O(n)$	$O(r)$
Storage cost	$O(n)$	$O(S)$	$O(\sqrt{n})$	$O(n)$	$O(n)$
Data availability	L	L	L	L	L
Data accuracy	N/A	L	L	L	N/A

Table 3.5: Comparison of data replication protocols not addressing ad hoc issues (Part 2)

	SAF, DAFN, DCG [57]	E-SAF, E-DAFN, E-DCG [58]	AL, GM [59]	AL+, GM+ [60]	Greedy-S, OTOO, RN [148]
Architecture	D	D	D	D	D
Read-only	Yes	No	Yes	No	Yes
Partition-aware	No	No	No	No	No
Energy-aware	No	No	No	No	No
Scalability	No	No	No	No	No
Robustness	No	No	No	No	No
Localized	No	No	No	No	No
Routing dependency	Any	Any	Any	Any	None
Replication cost	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Update cost	N/A	0	N/A	0	N/A
Query cost	$O(n+\sqrt{n})$	$O(n+\sqrt{n})$	$O(n+\sqrt{n})$	$O(n+\sqrt{n})$	1
Storage cost	$O(S)$	$O(S)$	$O(S)$	$O(S)$	$O(S)$
Data availability	L	L	L	L	L
Data accuracy	N/A	L	N/A	L	N/A

- *Read-only:* Does the replication protocol support only query operations?

- *Architecture:* The system architecture determines how the data items are deployed. There are two types of architecture: (1) *Centralized (denoted by C)*: the database is held by a server node and the replication process copies the whole database to a new server, or (2) *Distributed (denoted by D)*: each node holds its data items and the replication process copies one or some data items to another node.

- *Robustness:* A replication protocol is considered to be robust if it performs well not only under static environment but also under dynamic environment, i.e, the

Table 3.6: Features and performance of ERR

	Architecture	Read-only	Partition-aware	Energy-aware	Scalability	Robustness	Localized
ERR [60]	C	Yes	No	Yes	No	Yes	No

	Routing dependency	Replication cost	Update cost	Query cost	storage cost	Data availability	Data accuracy
ERR [60]	T	$O(n)$	N/A	$[1, O(n+\sqrt{n})]$	$O(n)$	MH	N/A

Table 3.7: Comparison of partition-aware data replication protocols

	Service coverage [142]	Hasupie [65]	Chen [26]	DAFN-S1, DAFN-S2, DCG-S1 [61]	DRAM [71]	Jorgic [77]
Architecture	C	C	D	D	D	C
Read-only	Yes	Yes	Yes	Yes	Yes	Yes
Partition-aware	Yes	Yes	Yes	Yes	Yes	Yes
Energy-aware	No	No	Yes	No	No	No
Scalability	No	No	No	No	No	No
Robustness	Yes	Yes	Yes	Yes	Yes	Yes
Localized	No	No	No	No	No	Yes
Routing dependency	None	T	P	None	None	None
Replication cost	1/group mobility	≥ 1	1/partition	$O(n)$	$O(n)$	≥ 1
Update cost	N/A	N/A	N/A	N/A	N/A	N/A
Query cost	$O(\sqrt{n})$	$O(n+\sqrt{n})$	$O(\sqrt{n})$	$O(n+\sqrt{n})$	N/A	N/A
Storage cost	$O(n)$	$O(n)$	N/A	$O(S)$	N/A	$O(n)$
Data availability	MH	MH	MH	MH	MH	MH
Data accuracy	N/A	N/A	N/A	N/A	N/A	N/A

data availability is not highly affected by topological changes caused by node mobility and node failures.

- *Localized:* A replication protocol is said to be localized if each node can make decision based only on the information from nodes within a constant hop distance.

- *Routing dependency:* Does the replication protocol depend on any particular routing protocol? If so, is it topology-based or position-based ?

Based on the above metrics an features, Tables 3.4, 3.5, 3.6, 3.7, 3.8, 3.9 and 3.10 compare the previously described data replication protocols. We summarize our notations in Table 3.3.

In the tables, the values for these metrics represent worst case behavior. We assume that data availability takes five qualitative values, which are: *Low (L)* if the replication protocol does not handle node mobility or node failures, *Medium-Low (ML)* if it handles either link failures or node failures, *Medium (M)* if it handles node and link failures, *Medium-High (MH)* if the protocol is partition-aware or energy-aware, and *High (H)* if it is partition-aware and energy-aware. Data accuracy takes

Table 3.8: Comparison of cluster-based data replication

	DHTR [150]	CDRA [154]
Architecture	C	D
Read-only	No	No
Partition-aware	No	No
Energy-aware	Yes	No
Scalability	No	No
Robustness	No	No
Localized	No	No
Routing dependency	None	None
Replication cost	M	$O(n)$
Update cost	$O(n+\sqrt{n})$	$O(2n)$
Query cost	$O(n+\sqrt{n})$	$O(n+\sqrt{n})$
Storage cost	(n)	N/A
Data availability	L	L
Data accuracy	L	L

Table 3.9: Comparison of location-based data replication

	GCLP [29]	GHT [120]	RR [125]
Architecture	D	D	D
Read-only	No	No	No
Partition-aware	No	No	No
Energy-aware	No	No	Yes
Scalability	Yes	No	No
Robustness	No	No	No
Localized	Yes	Yes	Yes
Routing dependency	P	P	P
Replication cost	n	$O(n)$	$O(n)$
Update cost	$O(\sqrt{n})$	$O(\sqrt{n})$	$O(\sqrt{n})$
Query cost	$O(\sqrt{n})$	$O(\sqrt{n})$	$O(\sqrt{n})$
Storage cost	(n)	$O(1)$	$O(\sqrt{n})$
Data availability	L	L	MH
Data accuracy	L	L	MH

its values based on the same criteria as the data availability metric. The routing dependency metric can take four values: (1) T or (2) P if the replication protocol utilizes a topology-based or a position-based routing protocol respectively, (3) *Any* if it does not matter which routing protocol is deployed, and (4) *None* if no routing protocol is used.

Table 3.4 and 3.5 show the performance and features of replication protocols not addressing the ad-hoc network issues. These protocols provide low data availability and data accuracy since they do not take the issues of network partitioning and battery exhaustion into consideration, which make them bad choices as replication protocols. The dissemination category [64, 67] incurs high overhead since each update packet is flooded in the whole network, which leads to the complexity of $O(n)$. Its unique advantage is that each nodes uses location information to flood data items.

Table 3.10: Comparison of location service protocols

	HIGH-GRADE [151]	GLS [91]	DLM [147]	SLURP [145]	SLALoM [28]	CRLS [130]
Architecture	D	D	D	D	D	D
Read-only	No	No	No	No	No	No
Partition-aware	No	No	No	No	No	No
Energy-aware	No	No	No	No	No	No
Scalability	Yes	Yes	Yes	Yes	Yes	Yes
Robustness	No	No	No	No	No	No
Localized	Yes	Yes	Yes	Yes	Yes	Yes
Routing dependency	P	P	P	P	P	P
Replication cost	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Update cost	$O(v \log n)$	$O(v\sqrt{n})$	$O(v\sqrt[3]{n})$	$O(v\sqrt{n})$	$O(v\sqrt[3]{n})$	$O(v\sqrt{n})$
Query cost	$O(\sqrt{n})$ (uniform) $O(\log n)$ (localized)	$O(\sqrt{n})$ (uniform) $O(\log n)$ (localized)	$O(\sqrt[3]{n})$ (both)	$O(\sqrt{n})$ (both)	$O(\sqrt[3]{n})$ (both)	$O(v\sqrt{n})$ (both)
Storage cost	$O(\log n)$	$O(\log n)$	$O(\sqrt[3]{n^2})$	$O(\sqrt{n})$	$O(\sqrt[3]{n})$	$O(\sqrt{n})$
Data availability	L	L	L	L	L	MH
Data accuracy	L	L	L	L	L	MH

The replication protocols based on quorum systems [82, 96, 92] are not robust since a failure of a single node can disable the whole system, which make them unsuitable for ad-hoc networks. In order to perform an update or a query operation in a quorum system, a constant number of nodes must be contacted. Each of these operations is preceded by a routing discovery procedure that needs a complexity of $O(n)$ packets to establish a route between a source node and one node of a quorum set. The update and the query cost of each message depends linearly on the diameter of the network and thus scales with $O(\sqrt{n})$. Therefore, the overall update and query cost grows as $O(Q(n+\sqrt{n}))$. In [96], due to the epidemic nature of PAN, each server that receives the *update* message, makes a write operation and chooses a number of other servers to whom it sends the update packet. In this manner, the update packet may be received by all the nodes; therefore, the complexity of PAN is of $O(n)$, which is superior than the quorum system of [82]. However, the update operation in PAN takes more time than the usual update operation used in quorum systems. In [92], the replication cost of the virtual backbone depends on the average node degree γ, and the value of r. It has the complexity of $O(n/\gamma^r)$ servers. Each servers stores the whole database, leading to the storage cost of $O(n)$. In addition the query packets are sent towards server that are at most r hops away from the querying nodes. However, it imposes on nodes to use ZRP [56] as an underlying routing protocol to forward the query packets.

The three methods: SAF, DAFN, DCG [57] and their extensions: E-SAF, E-DAFN, E-DCG [58], AL, GM [59], and AL+, GM+ [60] methods assume that the access frequencies to data items from each node are known, which make them unscalable for large ad-hoc networks. Moreover, they are not robust. For example, SAF, DAFN, DCG, E-SAF, E-DAFN, and E-DCG do not explain how a node should behave when the data item which it wants to allocate is not reachable. In addition, AL, GM, AL+, and GM+ do not handle the case of request failures due to unreachable nodes holding the original data items or replicas. In these protocols, each node needs to broadcast its access frequencies to data items. This operation has a complexity of $O(n)$. The route between the node holding the original or a replica and the requesting node is of $O(\sqrt{n})$, which leads to a query cost of $O(n + \sqrt{n})$. In [148], the authors claim that they can increase data availability by replicating data items that neighbor nodes have already not replicated. Although the query delay is low and is limited to one hop, data availability is not much improved, since the protocol considers only neighboring nodes and not larger groups.

Table 3.6 shows the performance and the features of ERR [136]. In ERR, the centralized server replicates a data item when the frequency of requests for the data item exceeds a threshold value. ERR tries to reduce the query cost by first requesting the data items from its neighbor nodes, If the request is not satisfied, it contacts the server. This operation leads to the complexity of $O(n + \sqrt{n})$. This performance complexity shows that ERR is not scalable. The data availability of ERR is medium-high, because replica servers are chosen based on their residual battery power and other capability parameters. However, ERR does not improve so much data availability because the new server replicas are chosen only among the requesting nodes that are reachable to the current server. Also, it does not consider the case of nodes that lose their connection to the server due to network partitioning.

Table 3.7 contrasts the protocols addressing the issue of network partitioning. These protocols are robust since they can react preemptively to topology changes. In addition, they achieve medium-high data availability and accuracy, since they only address the network partitioning issue.

The replication process in [142] is triggered when the server node predicts the occurrence of network partitioning by using the location information provided by a GPS system, which leads to a full network service coverage with a cost of one server per mobility group. Although full service coverage is achieved, the proposed protocol

suffers from the fact that client nodes need frequent accesses to at least one of the servers in order that the server nodes run the *sequential clustering algorithm*, which incurs an additional overhead. This protocol and DRAM [71] both assume that nodes only move according to the group mobility model, which significantly limits their implementations.

In [65], the replication is triggered when the client node is about to be in a separate network partition or when the QoS of the connection between the client and the server becomes bad. Contrary to [142], the server node's concern in [65] is to provide the service to the actual client node and not to the other non-connecting nodes, which results in low data availability. Although the algorithm does not depend on GPS, a client node has to store multiple disjoint paths to the same server, which represents a heavy load on it. Moreover, Hauspie's protocol generates a higher number of replica servers than that of [142].

In [26], based on the information provided by GPS, a node can predict the partitioning of the group of nodes. The advantage of Chen's protocol is that it can provide the lowest service cost (i.e., one server per partition). However, The authors have not given a definition for the group of nodes. One can assume that the whole connected network is also a group. In this case, to predict network partitioning, a single node needs to collect the location information of all the nodes, which makes this solution unscalable, and limits its implementation.

The three methods proposed by Hara [61] (i.e. DAFN-S1, DAFN-S2, DCG-S1) do not predict network partitioning, but they decrease its effects by eliminating duplicate replicas of data items between two nodes if the wireless link that connects them is stable. The decision to replicate is taken only at every relocation period. If the relocation period is shorter than the time needed for a certain link to move from an unstable sate to a failed state, DCG-S1 in this case can be considered to be partition-aware.

In [77], Hauspie et al. have proposed localized algorithms to detect nodes and links that could partition the network. A node makes a decision to determine critical nodes or links based on limited local knowledge, called the *k-hop knowledge*. The proposed algorithms may detect some nodes or links as critical although they may not be globally critical. Moreover, these algorithms are executed even if the network is in a stable state, which makes them costly in terms of message overhead.

Table 3.8 contrasts the cluster-based data replication protocols. The obvious drawback introduced by almost all clustering algorithms [150, 154] is the additional signaling overhead incurred in order to maintain the cluster structure. Maintenance of clusters with mobility would be extremely difficult, given the dynamic topology changes. Reconfiguration information must be propagated across the nodes. Cluster head changes can cause a rippling effect, where the role change of a leader in the higher level of hierarchy results in subsequent changes in lower leaders all the way to the bottom of the hierarchy. Due to these disadvantages, the replication protocols that are based on the clustering approach offer poor scalability. To send an update or a query packet, a topology-based routing protocol must be invoked, which incurs higher data packet delay and overhead. Thus, the communication complexity of an update or a query operation scales with $O(n + \sqrt{n})$: $O(n)$ to execute the route discovery procedure, and $O(\sqrt{n})$ to transmit the packet to its target.

Table 3.9 contrasts the location-based data replication protocols [135, 150, 125]. To send an update or a query packet, these protocols need a message complexity of $O(\sqrt{n})$, which makes them more scalable to large ad-hoc networks. In this category, it seems that *RR* protocol outperforms *GCLP* and *GHT* because it requires only approximate location information and accordingly is more robust to errors and imprecision in location than *GCLP* and *GHT*.

Table 3.10 shows the performance of the location services proposed in the literature. The performance results can be found in [151]. From this table, we can see that the message complexity of HIGH-GRADE increases logarithmically with the number of nodes. It outperforms the protocols belonging to its category. We can also see that CRLS [130] CRLS outperforms the other location services in terms of location availability location accuracy because the intersection of row and column is guaranteed providing that the network is not partitioned. On the other hand, hierarchical location services imposes on querying nodes to contact a chain of servers in a certain order so as to retrieve the queried node's exact location. If one of these servers is unreachable, the query operation cannot be performed.

Using a position-based routing protocol by a replication protocol is a sword with two edges. On one hand, it ignores the effect of topological changes, and hence the protocol is highly scalable. On the other hand, the location-based replication protocol cannot improve data availability since it does not deal with the issue of network partitioning.

As discussed above, the partition-aware and the location-based protocols seem to be the most promising data replication protocols. In addition, the energy-aware component can be easily integrated in both of the two categories.

To achieve both high data availability and scalability, we argue that the partition-aware replication protocols should be deployed in small-scale ad-hoc networks, while the location-based data replication protocols should be deployed in large-scale ad-hoc networks that have high node degree, i.e. strongly connected networks in order that the occurrence of network partitioning can be reduced.

More work remains to be done in order find the intersection domain, in which a partition prediction algorithm and a scalable data replication protocol can work together. We think that another way to achieve this goal is by developing a scalable partition prediction algorithm.

3.7 Conclusion

In this chapter, we have provided overviews of different replication protocols for ad-hoc mobile networks. While most of the current solutions are known to be either partition-aware or scalable, there has no considerable effort in the research of replication protocols that address all the MANET issues, that is, none of them is energy-aware, partition-aware as well as scalable. Development of a data replication protocol that considers all the three issues would be an attractive topic for future research.

Through the study of different data replication protocols, we can conclude that each protocol has advantages and disadvantages and has certain situations for which it exhibits good performance. We argue that the most promising categories for the design of a data replication protocol is the partition-aware and the location-based categories. The partition-aware protocols improve significantly data availability, but they are not scalable to large ad-hoc networks. On the other hand, the location-based data replication protocols are scalable but provide low data availability.

To achieve both high availability and scalability, a better compromise should be found between the two categories, or each class should be deployed in its suitable environment, i.e., the partition-aware protocols perform well in a small-scale network, whereas the location-based protocols perform well in a large-scale network.

Chapter 4

Partition prediction algorithm for service replication

Important network applications and services such as web servers, location information databases, key management and certification authorities are inherently centralized [142]. Network partitioning prevents the nodes that do not belong to the centralized server node's partition from accessing the service. Let us consider an example of a realistic scenario of an ad-hoc network with centralized service. A tour guide deployed by tourist information center may provide maps, pictures, history of attractive sites [149]. Some visitors may lose access to the tour guide when wireless links fail or when the network splits up into partitions. To ensure service accessibility to all nodes (i.e., a service instance is available in each network partition), we need to replicate the service in the future disjoint partitions before their disconnection from the server node. The issue that might arise in this case is how to ensure anywhere service availability without generating a large number of servers.

In this chapter, we propose a *partition prediction algorithm* [41, 36] based on the partition detection mechanism of *TORA* [109]. TORA has the advantage that its control messages are localized to a very small set of nodes near the occurrence of a topological change, and nodes maintain routing information about adjacent nodes. By using TORA's partition detection mechanism and a metric that calculates the residual lifetime of a wireless link, we can predict network partitioning rather than detecting its occurrence, and hence we can create in advance a service replica in the future separate partitions.

4.1 Related work

4.1.1 Overview of Malpani's algorithm

Malpani et al. [100] have adapted *TORA*(*Temporally Ordered Routing Algorithm*) [109] to elect a unique leader in each network partition. The leader node creates a *directed acyclic graph (DAG)*, which is a set of directed links rooted at that leader.

A 6−tuple, H_i =(lid_i, τ_i, oid_i, r_i, δ_i, i) is associated with each node. It represents the height of node i in the DAG. lid_i denotes the identifier of the node currently considered to be the leader of the network partition to which i belongs. The triple (τ_i, oid_i, r_i) denotes the *reference level*. A new reference level is started by node i if it loses its last outgoing link. τ_i is set to the time when this event occurs, oid_i is set to i, the originator of this reference level, and r_i is set to 0. r_i indicates an unreflected reference level. r can be changed to 1, indicating a reflected reference level, which is used for detecting partitions. When a new reference level is created, it is larger than any pre-existing reference level, since it is based on the current time. δ_i and i induce the directions on the links among all the nodes with the same reference level. Whenever i changes its height, it sends an *Update* message containing its new height to its neighbors.

The heights are compared lexicographically[1] (where $0 < 1 < 2 \ldots$ and $A < B < C \ldots$). Links are ordered from higher to lower heights. For each $j \in$ i's neighbors (N_i), the link (i, j) is marked *outgoing* if H_i is higher than H_j. Otherwise, it is marked *incoming*.

Figure 4.1(a) depicts a *DAG* constructed for the leader node F. Initially, node F sets its height to (F, -1, -1, -1,0, F). It then propagates an *Update* message in the network. When a node i receives such a message from node j, it sets its height (lid_j,0,0,0, $\delta_j + 1$, i).

The partition detection mechanism of TORA works as follows: each node i (other than the leader node) that has no outgoing links because of link failure, defines a new reference level, that is, (τ_i, oid_i, r_i)=(t, i, 0) and (δ_i, i)=(0, i), e.g., node D in Figure 4.1(a). Node i then reverses all its incoming links and broadcasts an *Update* message to the neighboring nodes. Upon receiving this message, each node that loses all its

[1]Given x and y two vectors. x is lexicographically higher than y if the leftmost non-zero entry of $(x - y)$ is positive. e.g., in figure 4.1(a), $H_A > H_B$ since $H_A - H_B = (0, 0, 0, 0, 1, A - B)$ and the leftmost non-zero entry 1 is positive.

outgoing links toward the leader node, performs links reversal in a localized manner (i.e., it decides to redirect its links by knowing only the heights of its neighbor nodes). It changes its height and broadcasts in turn an *Update* message to its neighbors. In this manner, the new reference level is propagated resulting in links reversal so that either the resulting graph is again leader-oriented or a network partitioning is detected. A node i can change its height under the following five different cases:

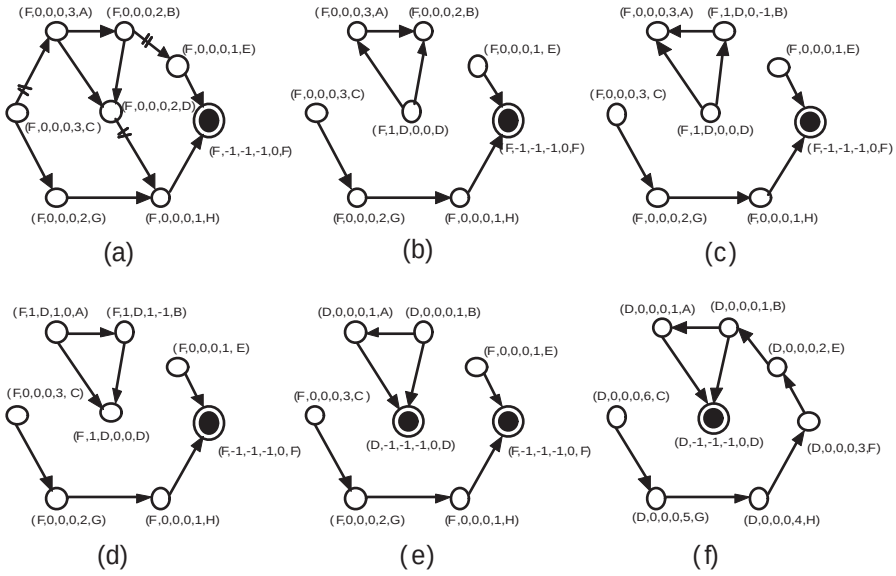


Figure 4.1: An execution of Malpani's algorithm

- *Case 1 (Propagate):* The node has no more outgoing link due to a link reversal following the receipt of an Update packet and the reference levels of its neighbors are not equal (e.g. node B in Figure 4.1(b)), It sets its reference level to the references level of its highest neighbor and sets δ_i to a value which is lower (-1) than the δ of all its neighbors with the maximum level. Formally:

$$(\tau_i, oid_i, r_i) = \max\{(\tau_j, oid_j, r_j) | j \in N_i\}$$

$$(\delta_i, i) = (\min\{\delta_j | j \in N_i / (\delta_j, oid_j, r_j) = (\delta_i, oid_i, r_i)\} - 1, i)$$

  The modification results in the graph in Figure 4.1(c).

- *Case 2 (Reflect):* The node has lost its outgoing links due to a link reversal

following the receipt of an update packet and the reference heights of the neighbors are equal with the reflection bit equal to 0, e.g. node A in Figure 4.1(c), then it reflects the reference level by setting r_i to 1 and δ_i to 0, resulting in the graph in Figure 4.1(d). It then propagates this reflected reference level back toward the node which originally defined the new reference level.

- *Case 3 (Detect):* The node has lost its outgoing links due to a link reversal following the receipt of an update packet and all of is neighbors have the same reflected reference level with $oid = i$, then i has detected a partition. The node that detects a partition (e.g., node D in Figure 4.1(d)) elects itself as the leader of the new partition, and sets its height to $(i, -1, -1, -1, 0, i)$. It then starts a DAG propagation that consists of diffusing this information to the nodes of the new partition. As illustrated in Figure 4.1(e), this propagation results in the creation of a DAG for node D.

- *Case 4 (Generate):* If all its neighbors have the same reflected reference level with an originator different from i, then i starts a new reference level. This situation happens if a link fails while the system is recovering from an earlier link failure. Because node i didn't define the new reference level itself this is not necessarily an indication of a partitioning of the component. So the node starts a new reference level. It sets its reference level to $(t, i, 0)$, and δ_i to 0, where t is the current time.

- *Case 5 (Merge):* The node receives an update packet from j such that $lid_j < lid_i$, then $H_i = (lid_j, \tau_j, oid_i, r_i, \delta_j + 1, i)$. In Figure 4.1(f), after the merging of two network partitions whose leaders are D and F respectively, D becomes the leader of the resulting partition.

4.1.2 Residual Link Lifetime Assessment

Nodes or links are said to be critical if their removal will split up the network into different separates partitions. Detecting such nodes and links can help to perform some data or service replication before the server becomes no longer reachable. To do so, depth first search algorithms (DFS) [46, 52, 134] are proposed. However, these algorithm have a high complexity and require knowledge of the topology of the whole network. Jorgic et al. [77] propose localized algorithms to detect critical nodes

and links. The algorithms consider that each node has a view of k-hop neighbors. However, these algorithms are executed even if the network is in a stable state, which represents a vain effort. The most appropriate choice we can do is to detect the critical nodes and links when they are about to fail (i.e., predict failures), and this can be only done by estimating the residual lifetime of wireless links.

There are basically two ways to predict the residual link lifetime between two neighboring nodes. The first method assumes knowledge of motion parameters of mobile nodes (e.g., location information, speed, direction, and transmission range) provided by GPS. The method uses distance measurements between mobile nodes and changes in their speed, to determine the duration of time these two nodes will remain connected. In [132], an approach that depends on the use of GPS and a free space propagation model is proposed, where the received signal strength solely depends on distance between two nodes. However, GPS is infeasible under many conditions. It is e.g., unavailable in indoor environments and it is not suitable for small devices due to its high power consumption.

The second method uses received signal power measurements [104]. It assumes that the sender signal power is constant. Received signal power samples are measured from packets received from a neighbor node. The power of the signal received at a node is inversely proportional to the distance the receiver is from the transmitter, raised to an exponent, i.e., $P_r = P_s/r^n$, where P_r is the power of the received signal, P_s is the power of the transmitted signal, r is the distance between the transmitter and the receiver nodes and n can be between 2 for free space radio propagation model and 4 for urban environments. By knowing radio propagation model, it is possible to derive the distance of separation between the two nodes from the formula of P_r. It is also assumed that the maximum speed of mobile nodes is known. Hence, it is possible to predict when the nodes will move out of transmission range of each other. If the received signal power falls below a threshold value, the receiver node considers that the link is likely to break. However, this method does not account for channel fading and multipath effects, which can cause sharp and sudden fluctuations in signal power, because it is highly dependent on the specific surrounding terrain.

In a realistic environment, all nodes in the network may not always have access to GPS or one or more do not carry GPS receivers. Moreover, they are not supposed to know the propagation conditions in which they operate. For these reasons, we use the approach presented in [128] for the calculation of the link's residual lifetime. It tries

to predict the time when the received signal strength falls below a critical threshold using a measured value of average change in received signal strength.

In [128], each node i broadcasts periodically a *beacon* message to its neighbors. We assume that *beacon* messages are sent at constant signal strength. We denote by $S_{ij}^k(t)$ the received signal strength that corresponds to the k^{th} beacon message sent by node j, and which is received by node i at time t . By keeping track of the last N values of $S_{ij}(t)$ and the corresponding time instances for the beacons received by node j. Node i calculates the rate of change of signal strength $V_{ij}^k(t)$ at time t as shown in equation (4.1), where $(t - t_{prev})$ denotes the time elapsed between the reception of two successive *beacon* messages.

$$V_{ij}^k(t) = \frac{S_{ij}^k(t) - S_{ij}^{k-1}(t_{prev})}{t - t_{prev}}(db/s) \tag{4.1}$$

Then, the *residual link lifetime* of a link *(i,j)* denoted by $\xi_{ij}(t)$, is calculated. The formula of $\chi_{ij}(t)$ is given in equation (4.2), such that S_R denotes the signal strength corresponding to the power range R of node i and below which the link *(i,j)* is considered to be failed.

$$\chi_{ij}(t) = \frac{S_{ij}^m(t) - S_R}{\frac{1}{N-1}\sum_{k=m-N+1}^{m} |V_{ij}^k(t)|}(s) \tag{4.2}$$

The main advantage of this approach is that it does not specify which mobility or propagation model is needed. Moreover, it does not make assumptions about mobile node's maximum speed or other motion parameters like in [51, 118, 119].

4.2 Pull-based Service Replication Protocol (PSRP)

4.2.1 Assumptions and definitions

We model the network as a graph $G = (U, V)$ where U represents the set of mobile nodes and V represents the set of edges. We make the following assumptions about the nodes and the network:

1. Communication links are bidirectional in order that links reversal can be executed.

2. Communication links are FIFO, i.e. messages are delivered in order over a link between two neighbors.

3. There is no message loss, no node crash.

4. Nodes have no information about their locations or propagation models.

We propose a service replication protocol called PSRP [36]. This protocol is slightly similar to the leader election algorithm proposed in [100], but it differs from [100] in two features. The first one is the condition that must be satisfied to trigger the algorithm into action (i.e., propagating the reference level), and the second one is the graph on which the algorithm is executed. Before presenting the service replication protocol, we must first define the following terms:

Definition 1. ***Active service state****: A node is called Active server if it performs all the operations related to satisfy the requests of client nodes.*

Definition 2. ***Passive service state****: A node is called Passive server if it does not perform the service operations but it stores the information and data required to start this service (e.g., data tuples in the case of database servers, task, configuration,...etc). It transits to the Active service state when it detects network partitioning.*

Definition 3. ***Regular state****: A node is called Regular if it is neither Active server nor Passive server. A regular node transits to the Passive service state when it predicts the occurrence of network partitioning.*

Definition 4. *A link (i,j) is called* ***weak link*** *if the predicate $P \equiv (\chi_{ij} < \chi_{th})$ holds true. Otherwise, it is called* ***strong link****. A path composed only of strong links is called* ***strong path****. Otherwise, it is called* ***weak path****.*

Definition 5. *A Node is called* ***stable node*** *if it has at least a strong path towards an active server. Otherwise it is called* ***unstable node****.*

Definition 6. *A subgraph induced by a set of connected stable nodes is called* ***stable subgraph****. A subgraph induced by a set of connected unstable nodes is called* ***unstable subgraph****.*

Figure 4.2 depicts the life cycle of a link (i, j). When a link (i, j) is formed, it enters the *link formation* state and transits to the *weak* state. At this state, the

link either transits to the *strong* state or to the *link failure* state. The *formed link* cannot be *strong* without passing through the *weak* state and a *strong link* cannot fail without being *weak* before.

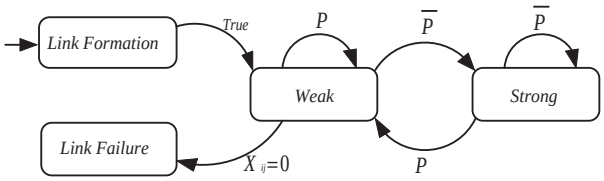


Figure 4.2: Life cycle of a wireless link

Figure 4.2 shows a state transition diagram for a node with three states. Initially, one node s is at the *Active service state* and the other nodes are at the *Regular state*. Transitions are labelled with triggering conditions stated in Algorithm 1, 2 and 3.

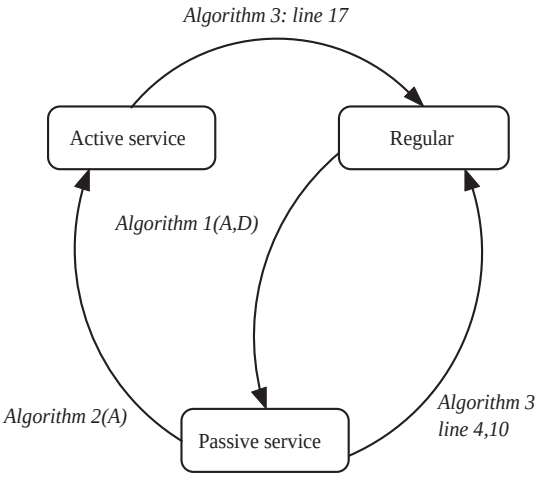


Figure 4.3: Node states transition diagram

4.2.2 Protocol description

The service replication protocol consists of three distributed algorithms executed by each mobile node, which are: (i) the *partition prediction algorithm* that predicts network partitioning and creates a passive service in the future separate partitions,

(ii) the *partition detection algorithm* that detects network partitioning and changes the state of the server in the new partition from passive to active and (iii) the *subgraph merging algorithm* that reduces the number of server nodes by determining which of them will terminate its service after the merging of two subgraphs.

For each node i, the service replication protocol maintains a set of variables. The variable (H_i) is the height of node i in the DAG, H_i=(s_i, τ_i, oid_i, r_i, δ_i, i), where s_i is the identifier of the *active server* in i's network partition. The variable (PH_i), PH_i=(Ps_i, $P\tau_i$, $Poid_i$, Pr_i, $P\delta_i$, i) represents the *Predict height* of node i. The *predict heights* of nodes are lexicographically ordered. They form another directed acyclic graph called $P{-}DAG$. The variable PN_i denotes the set of strong neighbors (i.e., nodes that have a strong connection with i). The variable $stability_i$ is set to $true$ if node i belongs to a stable subgraph and to $false$ if it belongs to an unstable subgraph. The variable $state_i$ represents the state of node i, which can take three values: *Active, Passive* and *Regular*.

For each node i in either a stable or an unstable subgraph, s_i denotes the current active server of node i. If a node i that belongs to an unstable subgraph, Ps_i is its current passive server. Two nodes i and j belong to different subgraphs if ($Ps_i \neq Ps_j$). A node i is called *active server* iff: ($i = s_i \wedge i = Ps_i$), *passive server* iff: ($i \neq s_i \wedge i = Ps_i$) and *regular node* iff: ($i \neq s_i \wedge i \neq Ps_i$). Events that occur in the network are defined to be the following:

- ***Event 1****: A node loses its last outgoing link due to a link failure.*
- ***Event 2****: A node gets a new link that joins two network partitions.*
- ***Event 3****: The last strong outgoing link becomes weak.*
- ***Event 4****: A node i gets a new strong link (i.e., a weak link becomes strong) that joins two subgraphs.*

The *partition detection algorithm* presented in [100] is triggered when *Event 1* or *Event 2* occurs and the *partition prediction algorithm* is triggered when *Event 3* or *Event 4* occurs.

4.2.3 Initialization

Initially, the centralized active server node begins the construction of the DAG and the $P{-}DAG$. It results in that (1) each node i has a directed path towards the centralized

server node and (2) $PH_i = H_i$. Whenever i's *Predict height* changes, node i sends an $P-Update$ message containing its new *Predict height* to each $j \in PN_i$. An example of the $P-DAG$ creation for the active server node F is shown in Figure 4.4(a). In Figure 4.4, the respective *Predict heights* are shown adjacent to each node. The active servers are marked by a circle and the passive server by a square. The strong and weak links are shown as solid and dotted lines respectively. The arrow on each wireless link points from the higher *Predict height* node to the lower *Predict height* node.

4.2.4 Partition prediction

Whenever a regular node i loses its last strong outgoing link towards an active server (Algorithm 1(A)), it considers that all the outgoing links are likely to fail soon. It initiates a warning by defining a new $P-$*reference level* (*Predicted reference level*). Unlike the *leader election algorithm* [100] that is executed on the graph G, the partition prediction algorithm is executed on the spanning subgraph $G' = (U, V')$, where

$$V' = V - \{all\ the\ weak\ links\ in\ the\ network\}$$

A node i that defines a new $P-$*reference level*, transmits then a $P-Update$ to its strong neighbors. Upon receiving this message, each node that loses all its strong outgoing links towards the active server node, changes its *Predict height* and transmits a $P-Update$ message to its strong incoming neighbors and so on, causing links reversal of the $P-DAG$. The new $P-$*reference level* is propagated over the strong links of the subgraph $G_i = (U_i, V_i)$ where:

$$\begin{cases} U_i = \{x / x = i \vee [i, x]\ is\ a\ strong\ path\} \\ V_i = \{(x, y) / x \in U_i \wedge (x, y) \in V'\} \end{cases}$$

Nodes change their *Predict height* in the same way like in cases 1, 2, 3, and 4 presented in Section 4.1.1 i.e., H_i and N_i are replaced by PH_i and PN_i respectively, and the sentence *"The node has lost its last outgoing link"* is replaced by *"The node has lost its last strong outgoing link"*. In Figure 4.4(a), when node D loses all its strong outgoing links, it propagates a new $P-$*reference level* to all $j \in PN_i$. In Figure 4.4(b), nodes A, and B change their *Predict heights* in response to the $P-$*reference level* propagation. Node B executes Case 2 of Section4.1.1, and propagates back a

reflected *P−reference level* toward node A by setting its Pr_B to 1. In figure 4.4(c), node A finds that all its strong neighbors have the same reflected*P−reference level* $(1, D, 1)$.

In Algorithm 1(B), when node i, which has started the new *P−reference level*, knows that all its strong neighbors have the same reflected reference level, it considers that a network partitioning may occur. Then, it downloads the service from its active server s_i, it transits from the *Regular state* to the *Passive service state*, and sets $stability_i$ to false . Node i updates its PH_i and propagates a *P−Update* in its unstable subgraph G_i. Upon receiving this information (Algorithm 1(C)), each node in G_i modifies its PH_i and sets its *stability* to false. Figure 4.4(d) depicts the system when node D knows that the failure of the actual weak links *(C, A)* and *(D, H)* will disconnect its subgraph $G_D = \{A, B, D\}$ from the active server node. In this case, it downloads a copy of the service from the active server F and initiates notifying other nodes which now belong to an unstable subgraph. The unstable subgraph G_D has exactly one *passive server* node (i.e., node D) and the stable subgraph G_F has exactly one *active server* node (i.e., node F).

Algorithm 1 Partition prediction algorithm at node i

Case A: When *Event 3 occurs*
1: **if** $PN_i = \phi$ **then**
2: ***download*** *the service from* s_i*;*
3: PH_i =(i, -1, -1, -1, 0, i);
4: $stability_i = false$*;*
5: $state_i$=*Passive;*
6: **else**
7: PH_i =(s_i, t, i, 0, 0, i); {*i starts a new P−reference level*}
8: **end if**

Case B: When *i predicts the occurrence of partitioning*
1: **download** *the service* from s_i;
2: PH_i =(i, -1, -1, -1, 0, i);
3: $stability_i = false$*;*
4: $state_i$=*Passive;*

Case C: When *i receives P−Update from j and* $Ps_i \neq Ps_j$
1: **if** ($stability_i = true \wedge stability_j = false \wedge (Poid_i = Ps_j \wedge Pr_i = 1)$) **then**
2: $stability_i = false$
3: PH_i =(Ps_j, 0, 0, 0, $P\delta_j + 1$, i);
4: **end if**

4.2.5 Partition detection

In Algorithm 2(A), when the passive server node i detects that it belongs to a network partition different from the active server node's partition, it transits to *Active service state* and sets $stability_i$ to true, then it starts providing the service and propagates this information to the the nodes of its stable subgraph G_i. Upon receiving this message (Algorithm 2(B)), each node in G_i sets its *stability* to true. Figure 4.4(e) depicts the system after the occurrence of network partitioning, node D starts providing the service and creates its DAG and $P{-}DAG$ for its new stable subgraph.

Algorithm 2 Partition Detection algorithm at node i

Case A: When *i detects the occurrence of partitioning*
1: **if** $state_i$=*Passive* **then**
2: $H_i = (i, -1, -1, -1, 0, i)$;
3: $PH_i = H_i$;
4: $stability_i = true$;
5: $state_i$=*Active;*
6: **end if**

Case B: When i receives *Update* from j and $s_i \neq s_j$
1: **if** $(stability_i = false \wedge stability_j = true)$ **then**
2: $H_i = (s_j, 0, 0, 0, \delta_j + 1, i)$;
3: $PH_i = H_i$;
4: $stability_i = true$
5: **end if**

4.2.6 Merging of two subgraphs

When a strong link (i, j) appears to combine two subgraphs G_i and G_j into one subgraph (Algorithm 3), three different cases can be distinguished:

1. *A strong link joins a stable subgraph and an unstable subgraph (Algorithm 3: line 4).* The resulting subgraph is stable and the passive server node in the unstable subgraph terminates its service.

2. *A strong link joins two unstable subgraphs (Algorithm 3: line 10).* The resulting subgraph is unstable and the passive server node in the unstable subgraph that has the greater identifier terminates its service.

3. *A strong link joins two stable subgraphs (Algorithm 3: line 17).* The resulting subgraph is stable and the active server node in the stable subgraph that has

the greater identifier terminates its service.

These three cases guarantee that after the merging of two subgraphs, each stable subgraph has a unique active server node and each unstable subgraph has a unique passive server node. For example, in Figure 4.4(f), after the creation of a new strong link *(B, E)* that merges two stable subgraphs G_D and G_F, node F that has the greatest identifier terminates its service, whereas D continues providing the service.

Algorithm 3 subgraph merging algorithm at node i

When *Event 4 occurs or (i receives PUPD from j and* $Ps_i \neq Ps_j$*)*
if *Event 4* **then**
  $PN_i = PN_i \cup \{j\}$;
end if
if $(stability_i = false \wedge stability_j = true)$ **then**
  $stability_i = true$;
  if $state_i$=*Passive* **then**
    $state_i$=*Regular;*
  end if
  PH_i =(Ps_j, 0, 0, 0, $P\delta_j + 1$, i);
else if $(stability_i = false \wedge stability_j = false)$ **then**
  if $Ps_i > Ps_j$ **then**
    if $state_i$=*Passive* **then**
      $state_i$=*Regular;*
    end if
    PH_i =(Ps_j, 0, 0, 0, $P\delta_j + 1$, i);
  end if
else if $(stability_i = true \wedge stability_j = true)$ **then**
  if $Ps_i > Ps_j$ **then**
    if $state_i$=*Active* **then**
      $state_i$=*Regular;*
    end if
    PH_i =(Ps_j, 0, 0, 0, $P\delta_j + 1$, i);
  end if
end if

4.2.7 Lower bound of the residual link lifetime

In order to ensure that the service will be reachable for nodes D, A, and B after the occurrence of network partitioning, we need to copy a service instance on one node of the future separate partition beforehand. For this purpose, we have to determine the lower-bound of the appropriate time to trigger the replication process. Node that starts a new *reference level* needs that the residual lifetime of the last strong outgoing link be sufficient to predict the partitioning and to download a copy of the service.

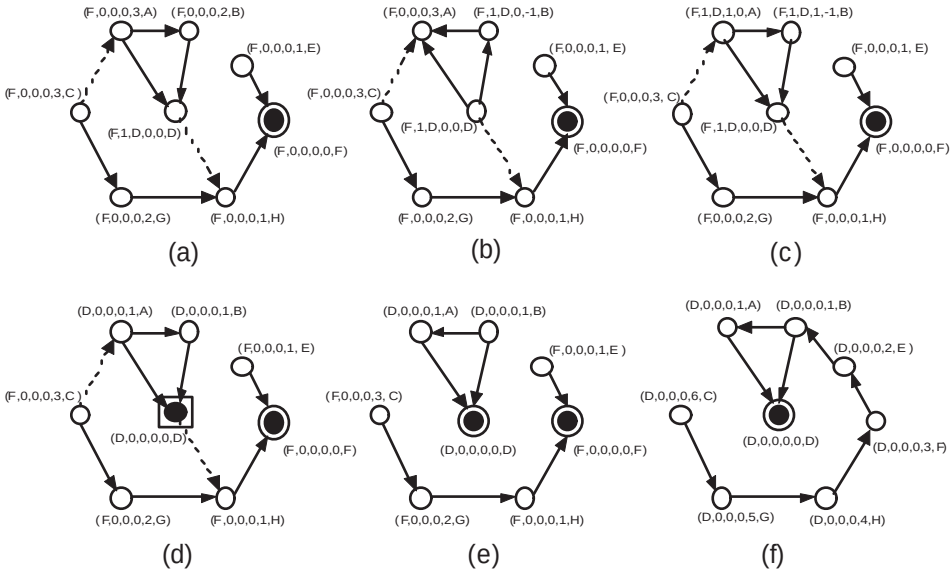


Figure 4.4: An execution of service replication protocol

Formally, the following predicate (χ_{th} >*time of service replication process+time to predict the partitioning*) must hold true. Let us consider that S denotes the size of data that need to be replicated and transmitted over an end−to−end wireless connection with a bandwidth of B bits/s. The service replication process is performed in $(\frac{S}{B})$ seconds. In [58], it is shown that at worst case, *TORA* needs $O(2l)$ units of time to detect a network partitioning, such as l is the length of the longest directed path in the network segment affected by a topological change. The partition prediction algorithm needs the same complexity of time to predict the partitioning. As a result, the lower-bound of χ_{th} is $(O(\frac{S}{B} + 2l))$ units of time.

4.2.8 Discussion

In practice, the received signal strength is largely dependent on actual radio conditions. Due to fading effects, it is subject to large fluctuations. The residual link life time estimation disregards the effects of path loss as well as the possibly strong fluctuations in signal strength caused by small scale fading effects.

According to equation (4.2), receiving a beacon packet with a dropped signal strength will decrease the residual link lifetime, since the rate of change $V_{ij}^{k}(t)$ is increased and the value of the numerator in equation (4.2) is low. Hence, the formula of $\xi_{ij}(t)$ presented in Section 2 can not give accurate values. In this case, the node

may trigger a false warning causing unnecessary *P−reference level* propagation, which may lead the partition prediction algorithm to make a false prediction and create an unnecessary passive server. However, the formula proposed in [128] mitigates the effect of channel fading and other transient interferences, because the trigger of a warning is based on a set of received beacon packets rather than a single packet, in order to verify that the signal strength drop was not due to fading. However, it cannot totally discards these effects.

4.3 Formal Verification of Prediction Property

We use linear temporal logic with past [86] as a formal tool to prove the correctness of the prediction property. In our proof, we use some temporal operators like : $\Box$ ('at every moment in the future'), $\Diamond$ ('eventually'), $\blacksquare$ ('at every moment in the past'), $\blacklozenge$ ('at some moment in the past') and $\mathcal{U}$ ('Until'). Before proving the correctness of the property, we first introduce some predicates employed by our proof.

- $Fail_{ij}$ (resp., $Form_{ij}$): is true as long as the link (i, j) does not exist (resp., exists).
- $Weak_{ij}$ (resp., $Strong_{ij}$): it evaluates to true if the link (i, j) is weak (resp, strong).
- $Fail_i$ (resp., $Weak_i$): it evaluates to true if node i has no outgoing links (resp., no strong outgoing links).
- $Detect_i$ (resp., $Predict_i$): it evaluates to true if node i detects (resp., predicts) network partitioning.

Our partition prediction algorithm ensures the following property: *whenever a network partitioning is detected by node i, then it has been predicted before by the same node.* More formally, this property is written as follows:

$$\Box(Detect_i \Rightarrow \blacklozenge Predict_i) \tag{4.3}$$

Proof. As depicted in Figure 4.2, whenever node i loses its link (i, j) at time t, then this link was weak at time t', where: $t - \xi_{th} < t' < t$. Formally, we have: $\Box(Fail_{ij} \Rightarrow \blacklozenge Weak_{ij})$.

Now we discuss the case of losing the last k outgoing links $(i, j_1), \ldots$, and (i, j_k) at $t_1, \ldots$, and t_k respectively. We assume that at time t_k, $Fail_i$ holds true following the loss of (i, j_k). Formally: $Fail_{ij_k} \Rightarrow Fail_i$

Node i propagates a new *reference level* in the spanning graph $H = (U, V - \{(i, j_1), \ldots, (i, j_k)\})$. Each outgoing link (i, j_m) that fails at time t_m $(1 < m < k)$, was weak at time t'_m such that: $t_m - \xi_{th} < t'_m < t_m$. At time t'_k, when node i losses its last strong outgoing link (i, j_k), the link becomes weak (as depicted in Figure 4.2), the other links $\{(i, j_1), \ldots, (i, j_{k-1})\}$ are either weak or failed. They cannot transit to strong state, because all these links at a latter time (t_k) have failed. Formally:

$$Fail_i \Rightarrow \blacklozenge[\forall m \in \{1, \ldots, k-1\} : (Weak_{ij_m} \vee Fail_{ij_m})] \tag{4.4}$$

$$\square(Fail_i \Rightarrow \blacklozenge(\bigwedge_{m=1}^{k-1} (Weak_{ij_m} \vee Fail_{ij_m}) \wedge Weak_{ij_k})) \tag{4.5}$$

By (4.4) and (4.5), the last outgoing link lost at time t_k, was the last strong outgoing link lost at time t'_k. Formally:

$$\square(Fail_i \Rightarrow \blacklozenge Weak_i) \tag{4.6}$$

A time t'_k, $Weak_{ij}$ holds true. So, node i propagates a new *P−reference level* in the spanning subgraph $H' = (U, V' - \{(i, j_1), \ldots, (i, j_k)\})$. As $V' \subseteq V$, we can conclude that $H' \subseteq H$.

If the propagation of the *reference level* at time t_k results in a detection of a new separate partition H_i, then by (4.6), the *P−reference level* was propagated in the subgraph H'_i at time $t'_k < t_k$. As $H'_i \subseteq H_i$ and H_i is a separate partition, then H'_i is also a separate partition. As a result, the algorithm predicted the network partitioning at time t' before its occurrence at time t. Thus, the prediction property stated in (4.3) is proved. □

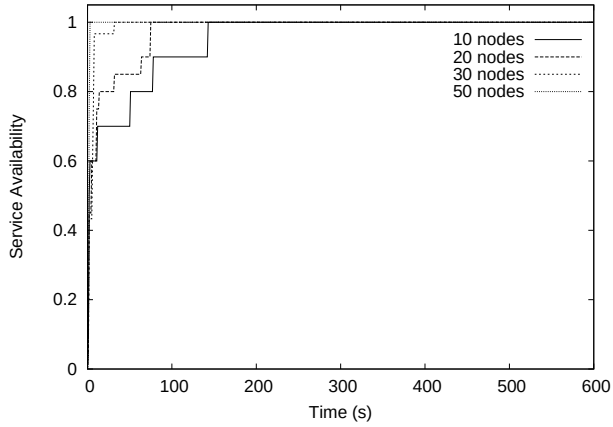


Figure 4.5: Service Availability

4.4 Simulation Results

4.4.1 Simulation scenarios

In [36], we have evaluated the performance of our protocol using GloMoSim simulator [153]. Our simulation environment is characterized by an area of 1000m×1000m, with random initial nodes' location, a random waypoint mobility model [74], in which each mobile node randomly selects a location with a random speed uniformly distributed between 0 and a certain maximum speed V_{max}, then it stays stationary during a pause time of 1 second before moving to a new random location. In the case of figure 4.5 and 4.7, V_{max} is set to 10 m/s. The transmission power is set to 5 dbm, which is equivalent to a transmission range of $198m$. We fix ξ_{th} at 1 second. Each simulation runs for 600 seconds. At the beginning of the simulation, one node is selected to be an active server.

To evaluate the performance of the service replication protocol, we define three metrics. Following is a brief discussion of these metrics:

- *Service availability:* The ratio between the number of nodes that can access the service and the total number of nodes in the network. A value of 1 indicates full network service coverage.

- *Service cost:* The average number of active and passive server copies per network partition. A network partition consists of one stable subgraph and multiple

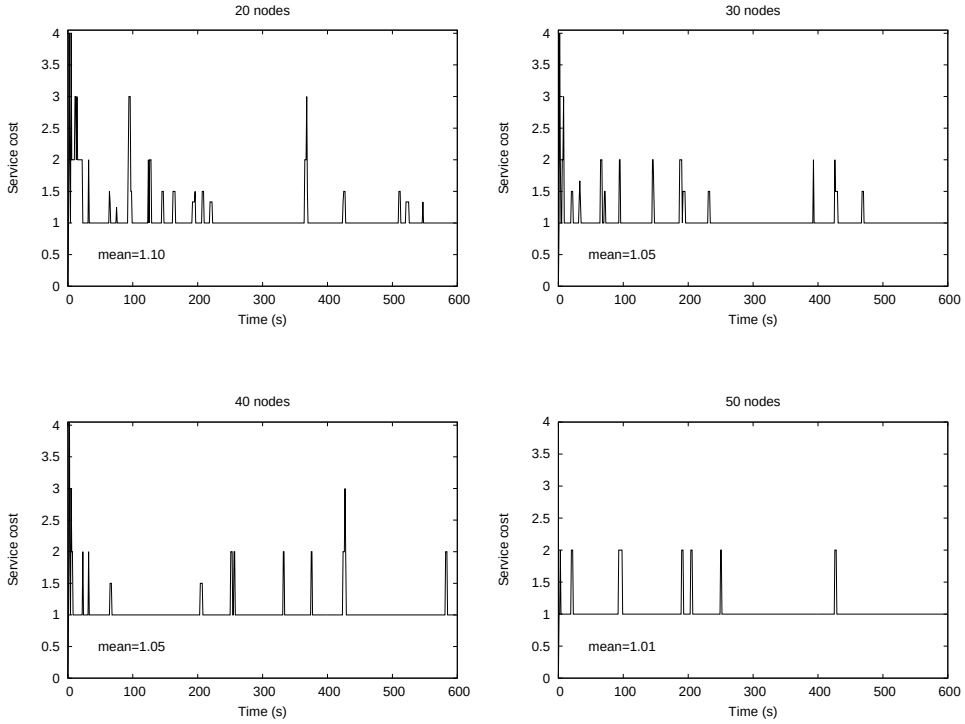


Figure 4.6: Service cost

unstable subgraphs with a cost of one active server per stable subgraph and one passive server per unstable subgraph. One active server per network partition represents the most optimal value. Ensuring full service coverage implies that the number of server nodes needs to be increased.

- *Prediction error:* Computes the number of times the partition prediction algorithm makes a false prediction. A node i makes a false prediction when $Detect_i$ event will not occur after the occurrence of $Predict_i$ event, because at least one of the outgoing weak link has become strong.

4.4.2 Service availability

We simulate four levels of network densities: low (10 nodes), low-medium (20 nodes), medium (30 nodes) and high (50 nodes). In Figure 4.5, we can notice that at the beginning of a simulation, the service availability is less than 1, because the nodes do not belong to the same network partition. As time progresses, the dynamic network

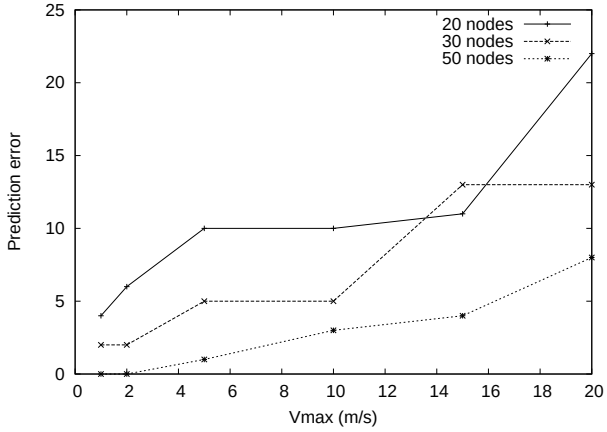


Figure 4.7: Prediction error

topology makes more nodes be accessible to an active server. Hence, the service availability is increased until it reaches a full service coverage. In a higher density network, the service availability converges quickly to 1 because the nodes are more likely to connect to a stable or an unstable subgraph.

4.4.3 Service cost

In Figure 4.6, the service cost is increased when a network partitioning is predicted, because an additional passive server is created for a new unstable subgraph. In the other hand, the service cost is decreased (i) when network partitioning occurs or (ii) when two subgraphs merge. Since in the first case, the node that was passive server become active for a new network partition and in the second case, one server is terminated. The service cost in low densities is higher than that of high densities, because as the density decreases, the occurrence of network partitioning becomes more frequent, and the number of events such that: partition prediction and partition detection, increases. Our service replication protocol incurs a mean service cost of 1.10 in the case of 20 nodes, 1.05 in the case of 30 and 40 nodes and 1.01 in the case of 50 nodes. From figure 4.6, we can see that the mean service cost results are very close to the optimal value.

4.4.4 Prediction error

Figure shows *prediction error* as a function of maximum speed V_{max} during a simulation run of 600 seconds. The prediction error is higher in the case of lower densities due to the increasing of the number of $Predict_i$ and $Detect_i$ events that occur in the network. In this case, the chances that $Detect_i$ will not occur after the occurrence of $Predect_i$ increases. From figure 4.7, we can see that the *predict error* increases when the density is decreased or when V_{max} is increased. The reason for this is that network partitioning occurs more frequently in these cases.

4.5 Conclusion

We have proposed a new mechanism to predict network partitioning, based on *TORA* partition detection mechanism and the residual link life time metric. The prediction of network partitioning leads to the creation of a passive server that will become an active server when a network partitioning occurs.

We have also proposed a service replication protocol, PSRP, that consists of three algorithms: a *partition prediction algorithm*, a *partition detection algorithm* and *subgraph merging algorithm*. The execution of the protocol creates a service replicas deployment scheme, which considers the future variations of network topology and minimizes wasted resources by making only one node to provide the service in a network partition.

Using linear time temporal logic, we have proved the prediction property. Our simulation results have shown that the service replication protocol ensures a full service coverage without incurring high service cost.

Chapter 5

Self-stabilizing Partition prediction algorithm for service replication

5.1 Introduction

A distributed system is self-stabilizing if it converges to a legitimate state in a finite number of steps regardless of the initial state, and the system remains in a legitimate state until another fault occurs. Many self-stabilizing algorithms have been designed to work on dynamic networks [10, 53, 139, 27, 44, 78]. However, they require that the network remains static during convergence, which make them unsuitable for ad hoc networks characterized by their highly dynamic topology. A new topological change brings the system into an unexpected state, and thus, the algorithms restart convergence to their legitimate states from scratch. Thus, they might never converge to their intended states in the presence of frequent and continuous topology changes. In [101], a self-stabilizing mutual exclusion protocol is proposed. The protocol works on a dynamic ring where nodes can join and leave the ring at any time. *Joins*, and *leaves* actions, which are considered as topological changes, modify the size of the ring, and brings the system into an illegitimate configuration. Each action of the self-stabilizing protocol is considered as a forward step to the convergence, and each topological change is considered as a backward step. By evaluating the degree of regression a topological change can bring about, the total number of the protocols steps required to complete the convergence despite the disturbance is evaluated.

In this chapter, we propose two algorithms: (1) a service coverage algorithm, which

is an enhanced version of PSRP [36], can only tolerate a single topology change, and (2) a self-stabilizing service coverage algorithm [40, 39] that can tolerate concurrent topology changes. One topological change means that a new topology change occurs only after the algorithm has terminated its execution triggered by a previous topology change, whereas concurrent topological changes mean that changes can occur at any time. The self-stabilizing service coverage algorithm can also ensure a high service availability, and converge to a legitimate state even if topological changes occur during the convergence time. This is achieved as follows: when multiple computations, which have the same goal, are executing during the same time period, we propose to stop newer computations in favor of the oldest one.

5.2 Service coverage algorithm

We propose the *service coverage algorithm*, a new replication method that can provide a continuous service availability for all mobile nodes. The service coverage algorithm is an enhanced version of PSRP presented in Chapter 4. It consists only of a partition prediction algorithm and a subgraph merging algorithm. In addition, it uses a new metric to estimate the residual lifetime of wireless links.

Intuitively, a link between two nodes i and j is broken either when both nodes are no longer within the transmission range of each other due to node mobility or one of which runs out of its battery power.

By knowing node i's remaining battery power E_i and its rate of energy dissipation R_i for every time period $\triangle t$, an ultra-conservative estimate of the residual node lifetime is derived as shown in equation 5.1.

$$\vartheta_i = \frac{E_i}{\max(R_i)}(s) \tag{5.1}$$

Each node j periodically broadcasts a *beacon* message containing its residual node lifetime ϑ_j, Upon receiving such a message from node j, node i first calculates d_{ij}, i.e. the distance separating it from its neighbor j. d_{ij} is estimated from the received signal power assuming that the radio propagation model is known. Node i then derives a conservative estimate of the residual link lifetime (the time when the neighbor j would move out of transmission range of node i) by assuming that both nodes are moving away even they are moving toward each other. The residual link lifetime, ξ_{ij}, is given

in equation (5.2), such that: R_{th} is the transmission range, i.e. the maximum distance between a sender and a receiver for successful packet reception, and above which the link (i, j) is considered to be failed, and V_{max} is the maximum node velocity.

$$\xi_{ij} = \frac{|d_{ij} - R_{th}|}{V_{max}}(s) \tag{5.2}$$

Therefore, each node i estimates the residual node-link lifetime by the following equation:

$$\chi_{ij} = \min(\vartheta_i, \vartheta_j, \xi_{ij}) \tag{5.3}$$

Then, it checks if the predicate $P \equiv (\chi_{ij} - \Delta t < \chi_{th})$ holds true. If so, node i considers that the link (i, j) is likely to break, and hence the link is called *weak*. Otherwise, it is called *strong*. Δt denotes the time period between successive beacon transmissions, and χ_{th} denotes the lower-bound of the residual node-link lifetime at which node i have to generate a warning and perform preemptive actions like service and data replication. The node for which the predicate $Q \equiv (\vartheta_i - \Delta t < \chi_{th})$ holds true is called *short-lived node*. Otherwise, it is called *long-lived node*. A path which is composed only of strong links is called ***strong path***. Otherwise, it is called ***weak path***. The connected subgraph obtained after removing the short-lived nodes and the weak links is called *long-lived connected component (LLCC)*[1].

If Δt is too large, the link may be broken during that interval before the node captures its weakness, and if it is too low, it may incur a high overhead. So, Δt must satisfy the constraints: $\Delta t < \chi_{th}$. We use Δt in the predicate P because when χ_{ij} falls bellow χ_{th}, there may be no time left for the preemptive actions.

In the service coverage algorithm, when a node predicts network partitioning, it becomes an active server, and creates a DAG for its new LLCC. In this chapter, we use the terms: height and reference level instead of P-height and P-reference level respectively.

5.3 Handling concurrent topological changes

As explained in chapter 4, to predict network partitioning, a reference level needs to traverse the entire new LLCC. When multiple reference levels are triggered during the

[1]Throughout this chapter, the terms subgraph, LLCC and component are used interchangeably.

same time interval, the most recent reference level stops the propagation of ongoing reference levels it collides with.

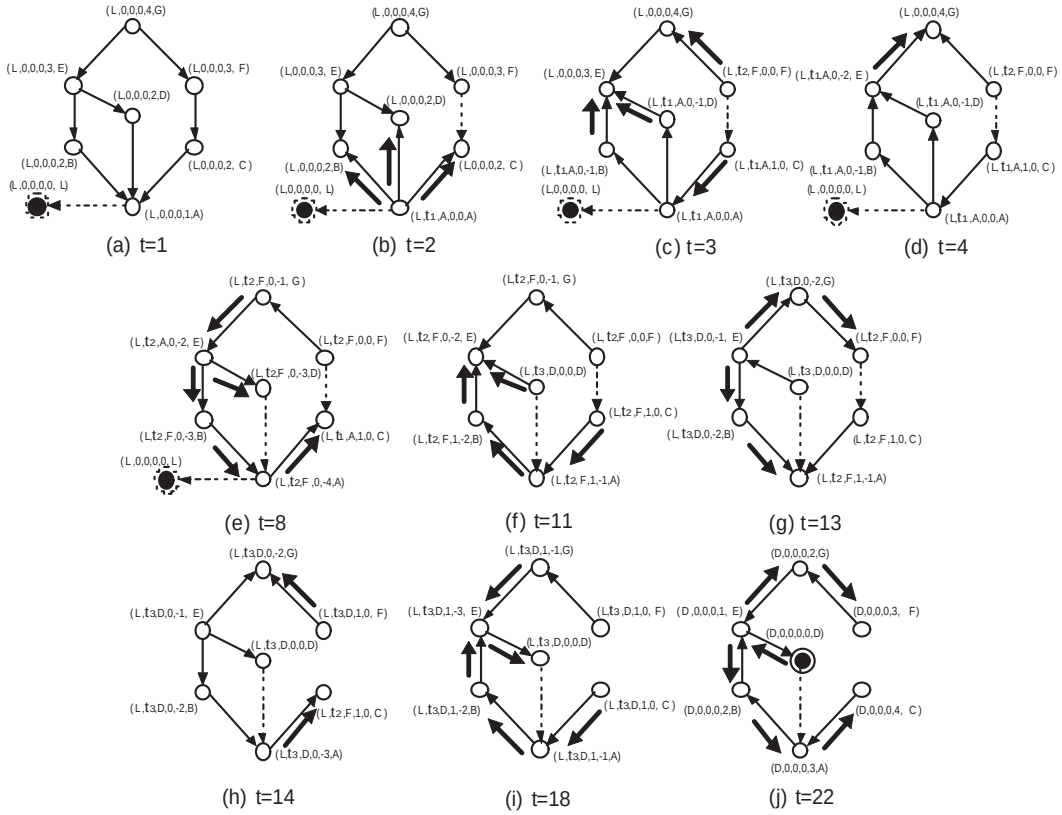


Figure 5.1: An example of executing the service coverage algorithm in case of concurrent topological changes

Figure 5.1 shows an example of executing the service coverage algorithm in the presence of concurrent topological changes. In figure 5.1(a), node A loses at time $t_1 = 1$ its last strong outgoing link toward its server node L. In figure 5.1(b), node A defines a new reference level (i.e. $ref_1 = (t_1, A, 0)$). The new reference level is now higher than that of the neighbors, so the update message has as effect the reversal of the links to B, C, and D. At time $t_2 = 2$, node F loses its last strong outgoing link. Node F defines a new reference level (i.e. $ref_2 = (t_2, F, 0)$), and broadcasts an update message to its neighbors, which results in reversing the link (G, F) as shown in figure 5.1(c). In the figure, nodes B, and D execute *case 1*[2] and node C executes *case 2* in

[2]In this chapter, case 1, 2, 3, and 4 are those defined in section 4.2.4

response to ref_1 propagation. In figure 5.1(d), node E changes its height according to *case 1*. In the figure, node G loses its last outgoing link following reception of ref_1. So, it executes *case 1* and sets its reference level to the largest among all its neighbors (i.e. ref_2), as depicted in figure 5.1(e). Figure 5.1(e) shows the propagation of ref_2 in the partitioned component. At time $t_3 = 8$, node D loses its last strong outgoing link (D, A). As shown in figure 5.1(f), node D defines a new reference level (i.e. $ref_3 = (t_3, D, 0)$), and sends an update message to its neighbor E, which has the effect of reversing the link to E. This figure also shows the propagation of the reflected reference level of ref_2 in the network until it reaches node E. In figure 5.1(g), as $t_3 > t_2$, node E sets its reference level to ref_3. ref_3 is then propagated until it reaches node F. In figure 5.1(h), F executes *case 2*, sets its reference level to $(t_3, D, 1)$, and propagates the reflected ref_3 back toward node D. In figure 5.1(i), node C sets its reference level to $(t_3, D, 1)$. The reflected reference level of ref_3 is propagated back toward node D. At time $t = 18$, node D finds that all its neighbors have the same reflected reference level with $oid = D$, so it has predicted network partitioning. Then, it executes *case 3*, and elects itself as server and propagates its height in the new component, which results in the creation of a *D-DAG*, as shown in figure 5.1(j).

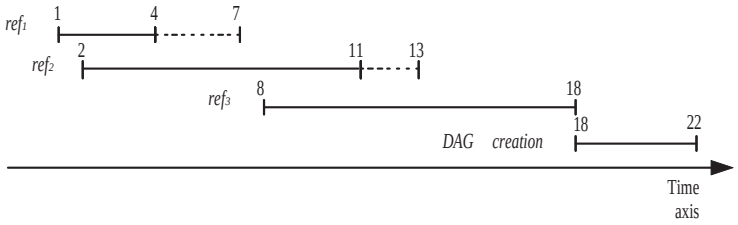


Figure 5.2: Time diagram of the service coverage algorithm execution

A new reference level is propagated outward from the point of the original event (re-directing links in order to re-establish routes to the server). This propagation will only extend through nodes which (as a result of the initial link weakness) have lost all strong routes to the server. To predict network partitioning, a reference level must pass through all nodes of the component in two phases: (i) forward phase, in which nodes propagate the reference level with $r = 0$, and backward phase, in which nodes propagate the reference level with $r = 1$. This leads to a time complexity of $O(2d)$,

where d is the diameter of the network component.

Figure 5.2 shows the execution of the example presented in figure 5.1 on a time axis. Solids lines represent the time intervals during which the reference levels are executed. Dotted lines represents the remaining time required for the algorithm to complete its execution if no further link weaknesses occur.

The reference level ref_1 was started at time $t_1 = 1$ (see figure 5.1(a)), and it was stopped by node G at time $t = 4$ (see figure 5.1(d)) due to the existence of ref_2 that was started at time $t_2 = 2$. If ref_2 was not occurred, the algorithm would predict network partitioning at time $t = 7$, since the diameter of the component at time $t = 1$ was $d = 3$. In the same way, the ref_2 propagation, which was supposed to terminate at time $t = 13$, is stopped by ref_3 at time $t = 11$ (figure 5.1(f)). The ref_3 propagation is terminated at time $t = 18$. Node D, which predicts network partitioning, creates a new DAG that takes 4 units of time to terminate.

In case of a single topological change, the service coverage algorithm needs $O(2d)$ time units to predict partitioning. From the example, we can conclude that each new reference level spoils all the efforts the algorithm made before to predict network partitioning. The algorithm will not terminate if reference level generations do not stop. So, the execution time of the service coverage algorithm may be infinite, i.e., nodes can be without a server for undetermined time. In this case, determining a lower-bound for the residual node-link lifetime χ_{th} becomes a nonsense, and hence network partitioning might never be predicted or it is predicted after it is too late.

As we can see, the service coverage algorithm is inefficient in ad hoc mobile networks because it converges to a legitimate state only if the topological changes stop. When a new topological change occurs before completing the convergence, the state immediately after the change is regarded as an arbitrary one. As ad hoc networks are subject to frequent and unpredictable topological changes, the algorithm might never converge to its intended behavior forever, and thus it is quite inefficient. It converges to a legitimate state only if topological changes are prohibited during convergence time.

5.4 Basic concepts

The state of a node is defined by the values of its local variables. A configuration of a distributed system is an instance of the states of its nodes. The set of configurations

of the network is denoted as $\mathcal{C}$. Node actions change the global system configuration. Moreover, several node actions may occur at the same time. The actions of each node i are of the form: $< guard_i > \rightarrow < command_i >$. The guard $< guard_i >$ of each action is a boolean expression on the state of i and its neighborhood. $< command_i >$ represents a list of assignment statements and primitives such as: broadcast messages. We assume that the guarded action can be atomically executed: evaluation of the guard and execution of the statement are executed in one atomic action.

Definition 7. ***Legitimate configuration of service coverage*** *Configuration σ is a legitimate configuration of the service coverage problem for a long-lived connected component C iff $\forall i \in C \exists l : lid_i = l$ and C is a directed acyclic graph with every long-lived node in C except for the server l has a directed strong path to l.*

Definition 8. ***self-stabilizing service coverage algorithm*** *The service coverage algorithm is self-stabilizing if any execution starting from an arbitrary configuration eventually reaches a legitimate configuration (convergence).*

Definition 9. *A node in a LLCC is said to be uncertain if it loses all its strong outgoing links either due to links weakness or links reversal, because that node still does not know if a new route toward the server is found. Otherwise it is said to be certain.*

Definition 10. *The subgraph induced by a set of connected certain nodes is called the certain subgraph. Otherwise, it is called the uncertain subgraph.*

Definition 11. *A node in an uncertain subgraph which is adjacent to any node in a certain subgraph is called a frontier node, and a node in a certain subgraph which is adjacent to a frontier node is called a border node.*

5.4.1 Time interval-based computations

In distributed systems, computations have durations. They can be specified by the instants at which they begin and end. In temporal logic, interval-based temporal logics are more expressive than instant-based ones since they are capable of describing events that occur in the system in time intervals. We could define an interval as an ordered pair $[t, u]$ of instants, where $t < u$. If $I = [t, u]$, we write *Begin(I)=t* and *End(I)=u*. This notation allows us to refer to the instants marking the beginning and

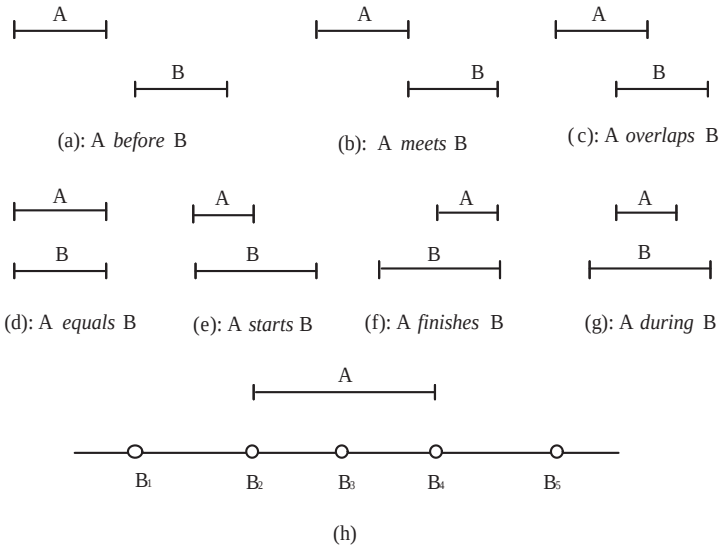


Figure 5.3: Possible relationship between intervals and instants

the end to any interval we refer to. A single time instant t can also be written as an interval $[t, t]$.

Allen [2] has defined 13 basic binary relations between time intervals, six of which are inverses of the other six: before and after, overlaps and overlapped-by, starts and started-by, finishes and finished-by, during and contains, meets and met-by, and equals (see Figure 5.3).

Two time intervals A and B *meet* (or B *met-by* A) if and only if A precedes B, yet there is no period between A and B, and A and B do not overlap [3]. Let *Meets(A,B)* be a predicate that evaluates to true if A *meets* B. The predicate *Before(A,B)* (or *After(B,A)*) holds true if there exists another period that spans the time between them. Two intervals A and B are disjoint if they do not overlap in any way. The predicate *Overlaps(A,B)*(or *overlapped-by(B,A)*) holds true if A starts before B, and B ends after A. The predicate *Equals(A,B)* holds true if A and B both starts and ends at the same time. The predicate *During(A,B)* or *Contains(B,A)* holds true if A has a later starting point and an earlier ending point than B. The predicate *Starts(A,B)* (or *Started-by(B,A)*) holds true if A has the same starting point as B, but it is contained in B. The predicate *Finishes(A,B)* (or *Finished-by(B,A)*) holds true if A has the same ending point as B, but it is contained in B.

After presenting the relation between intervals, we now need to define the relations that exist between an instant t and an interval I. The predicate *Precedes(t,I)* and *Follows(t,I)* hold true if $t < Begin(I)$ and $t > End(I)$ respectively. The predicate *Divides(t,I)* holds true if $Begin(I) < t < End(I)$. *Limit(t,I)* holds true if either $t = Begin(I)$ or $t = End(I)$.

5.5 Self-stabilizing service coverage algorithm

5.5.1 Basic idea

As shown in section 5.3, the service coverage algorithm does not stabilize if the generation of reference levels does not stop. Let us consider the situation where multiple reference levels are propagating during the same time period. In *Case 1*, the most recent reference level has always a higher priority than the others. In addition, *Case 4* chooses to generate a new reference level because it does not know if the network partitioning is about to occur or not. Also, the subgraph merging algorithm does not define a time-based order on the different DAG propagations that occur in a component. When a new topological change occurs before completing the convergence, the algorithm restarts convergence to its legitimate state from scratch. These situations raise the question of how the algorithm could stabilize in the presence of non-stop concurrent topology changes.

As all the reference level propagations, which are triggered in response to links weakness, have the same goal (i.e., checking if the component will be separated from the server node), we can intuitively propose to change the criterion used to order reference levels and consider the oldest reference level as the highest priority instead of the most recent one. A reference level ref_j is higher than another ref_i iff $\tau_j < \tau_i$. Roughly speaking, a reference level will be stopped when it collides with an older one. The less fresh propagation criterion incurs fewer message overhead and it is closer to completion than the criterion adopted by the service coverage algorithm. However, this approach is inefficient since we can make false decisions about network partitioning. Moreover, we cannot distinguish between concurrent computations and disjoint ones. Therefore, it is not sufficient to only know the time at which the reference level was generated, because we cannot determine if two reference levels are still propagating or one of which has been already terminated. So, we propose that

each node records its knowledge about the age of the reference level it meets, i.e., the time when the reference level was started and the time when the node receives this reference level for the last time. Hence, each node can decide if two reference levels are disjoint or concurrent. We changes *Case 1* as follows: If reference levels are disjoint, the node stops propagating older reference levels. Otherwise, it stops propagating the newer reference levels in favor of the oldest one. In case of merge of components, we also propose to stop DAG propagation associated with newer servers in favor of the oldest server.

5.5.2 Data structure

Each long-lived node i maintains two variables: the partition index and the height. The partition index (denoted by PI_i), is a 3-tuple $(Certain_i, Tc_i, lid_i)$. lid_i is the identifier of the node considered to be the server of node i. $Certain_i$, a boolean variable, whose value is 1 if node i is in a certain state, and 0 otherwise. Tc_i denotes the time at which node lid_i has started the creation of its DAG. A new partition index is started by node i when it predicts network partitioning. The height $H_i = (ERL_i, \delta_i, id_i)$, where ERL_i is the extended reference level. This latter is a 3-tuple $([Tb_i, Te_i], oid_i, r_i)$. The triple (Tb_i, oid_i, r_i) is still called a reference level and it is denoted by RL_i. The variable $[Tb_i, Te_i]$ is called the reference level interval of i (denoted by RLI_i), is the knowledge of node i about the time period during which the reference level is propagating. The definitions of oid_i, r_i, δ_i and id_i are still unchanged.

5.5.3 Ordering of time intervals and links orientation

Definition 12. *Two time intervals I and J intersect iff: $I \cap J \neq \emptyset$.*

Definition 13. *Two intervals I and J α-intersect, and we write $I \overset{\alpha}{\bigcap} J$ if there exists a chain of l intersections such that the following statement holds true.*

$$(I \cap I_1) \wedge (I_1 \cap I_2) \wedge \cdots \wedge (I_{l-2} \cap I_{l-1}) \wedge (I_{l-1} \cap J)$$

It is easy to show that the binary relation $\overset{\alpha}{\bigcap}$ on a set S of time intervals is an equivalence relation.

Definition 14. *For the equivalence relation* $\overset{\alpha}{\bigcap}$*, the set of elements of* S *that are related to a time interval, say* a*, of* S *is called the equivalence class of* a *and it is denoted by* S_a*, such that* $S_a = \{x \in S : x = a \vee a \overset{\alpha}{\bigcap} x\}$*. The set of equivalence classes of the equivalence relation* $\overset{\alpha}{\bigcap}$ *on a set* S *form a partition of* S*.*

Definition 15. $\{S_{I_1}, S_{I_2}, \cdots, S_{I_n}\}$ *is a partition of* S*, if and only if: (1)* $S_{I_i} \neq \emptyset, 1 \leq i \leq n$*, (2)* $S_{I_i} \cap S_{I_j} = \emptyset$*, if* $S_{I_i} \neq S_{I_j}, 1 \leq i, j \leq n$*, and (3)* $\bigcup_{i=1}^{n} S_{I_i} = S$*.*

Definition 16. *An equivalence class* S_a *is said to be more recent than* S_b *iff:* $\forall I \in S_a, \forall J \in S_b : End(J) < Begin(I)$*.*

Definition 17. *Let us consider two time intervals* I *and* J*, both belonging to the same equivalence class.* I *is said to be older than* J *if the following holds:* $(Meets(I,J) \vee Overlaps(I,J) \vee Finished-by(I,J) \vee Contains(I,J) \vee Limit(J,I) \vee Divides(J,I))$

Definition 18. *Let us consider two time intervals* I *and* J*, both belonging to the same equivalence class.* I *is said to be older than* J *if the following holds:* $(Meets(I,J) \vee Overlaps(I,J) \vee Finished-by(I,J) \vee Contains(I,J) \vee Limit(J,I) \vee Divides(J,I))$

In the directed acyclic graph, a link (i, j) is oriented from i to j according to the following conditions.

1. If nodes i and j are both certain and $\delta_i > \delta_j$ (i.e., (D, A) in figure 5.4(a)).
2. If both nodes are uncertain, we check their respective RLI:
   - If S_{RLI_i} is more recent than S_{RLI_j}, i.e., $After(RLI_i, RLI_j)$ holds true (e.g., (B, E) in figure 5.4(c)).
   - If RLI_i and RLI_j belong to the same equivalence class, and RLI_i is older than RLI_j (e.g., (G, F) in figure 5.4(e)).
   - If RLI_i and RLI_j are not related to the same reference level, and *Started-by* (RLI_i, RLI_j) holds true.
   - If RLI_i and RLI_j are related to the same reference level, and $(r_i = 1 \wedge r_j = 0)$ (e.g., (E, B) in figure 5.4(f)).

- If RLI_i and RLI_j are related to the same reference level, and $(r_i = r_j)$, and *Start(RLI_i,RLI_j)* holds true (e.g., (B, E) in figure 5.4(d), and (F, G) in figure 5.4(f)).
- If *Equals(RLI_i,RLI_j)* holds true and $(oid_i, r_i, \delta_i, id_i) > (oid_j, r_j, \delta_j, id_j)$.

3. If node i is uncertain and node j is certain (e.g., (A, D) in figure 5.4(b)).

5.5.4 Initialization

Initially, each long-lived node i whose hight is null (i.e. $H_i = ([-,-],-,-,-,i)$) can start the construction of a DAG. It sets PI_i to $(1, t, i)$ and H_i to $([0,0],0,0,0,i)$, where t is the current local time. Then, it broadcasts a *CreateDag(PI_i, H_i)* message to its strong neighbors. Upon receiving such a message, each node j whose height is null, sets PI_j to PI_i, H_j to $([0,0], oid_i, r_i, \delta_i + 1, j)$. Multiple DAG constructions can be triggered at the same time. DAG construction is depicted in Algorithm 4.

Algorithm 4 DAG Construction

Action 1: *(i wants to be a server* $\wedge H_i = null) \rightarrow$
1: $PI_i := (1, t, i)$; $H_i := ([0,0], 0, 0, 0, i)$;
2: *broadcast CreateDag(PI_i, H_i)*;

Action 2: *(i receives Creation(PI_j, H_j)* $\wedge H_i = null) \rightarrow$
1: $PI_i = PI_j$;
2: $H_i := ([0,0], oid_j, r_j, \delta_j + 1, i)$;
3: *broadcast CreateDag(PI_i, H_i)*;

5.5.5 Self-stabilizing partition prediction algorithm

Algorithm 5 consists of two actions. *Action 3* shows the reaction of certain and uncertain node due to links weakness, and *Action 4* shows the instructions performed by a long-lived node in response to links reversal.

Contrary to the algorithm proposed in Section 5.2, we start a new reference level only if a certain node loses all its strong outgoing links. When an uncertain node loses all its outgoing link, it does not need to generate a new reference level to propagate through uncertain subgraphs, which have been already explored by previous reference levels. Instead, it selects one of the reference levels to continue its propagation. This choice avoids incurring more effort in terms of of message overhead and time. It helps to quickly detect network partitioning if it occurs.

When a certain node i loses its last strong outgoing link, it creates a new reference level. It sets $Certain_i$ to 0 and ERL_i to $([t,t],i,0)$. Then, it broadcasts *Failure*(PI_i, H_i) to its neighbors.

When an uncertain node i loses its last strong outgoing link, it invokes the *HandleUncertainReverse* procedure. The procedure distinguishes four conditions. First, if i's strong neighbors do not all have the same reference level, it sets RL_i to the maximum reference level among its neighbors. It calculate using the function $\max_{\succ}$ the reference level interval (RLI) with the highest priority. This function determines the most recent equivalence class S, and then it returns the reference level with the oldest RLI among the intervals belonging to S. Second, if all strong neighbors have the same unreflected reference level, it reflects this reference level by setting r_i to 1, it means that the strong link from which it is supposed to receive a reflected reference level, is now weak. Third, if all strong neighbors have the same reflected reference level with an originator other than i, (i.e. no strong path between i and the originator of the reference level is available). This situation is likely to happen when a new LLCC is created in the network component where the reflected reference level is still propagating. In this case, it elects itself a server and propagates a *CreateWeakDag*$(PI_i, H_i, old(lid_i), old(Tc_i))$ message. This propagation constructs what we call a weak DAG. The weak DAG will be explained later in this paper. Fourth, if all strong neighbors have the same reflected reference level with i as an originator, it predicts network partitioning and starts creating a new DAG.

In *Action 4*, when node i receives the *Failure* message from its strong neighbor j (i.e. a reference level is propagating), it checks if it still has outgoing links. If so, this reference level will be stopped. Otherwise, it invokes the *HandleUncertainReverse* procedure.

Figure 5.4 shows an example of executing the proposed self-stabilizing service coverage algorithm on the network of figure 5.2. In figure 5.4(a), node A loses at time $t = 1$ its last strong outgoing link toward its server L. In figure 5.4(b), node A generates the reference level, $ref_1 = (1, A, 0)$. At time $t = 2$, node F defines a new reference level, $ref_2 = (t_2, F, 0)$, which results in reversing the link (G, F) as shown in figure 5.4(c). In the figure, nodes B, and D execute *condition 1* and node C executes *condition 2*. In figure 5.4(d), node E changes its height according to *condition 1*. In the figure, node G loses its last strong outgoing link following reception of ref_1. So, it executes *condition 1* and sets its reference level to the one with the highest priority

among its strong neighbors (i.e. ref_1), as depicted in figure 5.4(e). Figure 5.4(f) shows the propagation of the reflected reference level ref_1 toward node A. All nodes i such that: $RL_i = ref_2$, set their reference levels to ref_1. In this case, ref_2 is said to be deleted. In figure 5.4(f), the uncertain node D loses at time $t_3 = 8$ its strong last outgoing link (D, A). Node D executes *condition 3*, and creates a weak DAG, as shown in figure 5.4(g). In the figure, node A finds that all its neighbors have the same reflected reference level with $oid = A$, so it has predicted the partition. Then, it executes *condition 4*, and elects itself as a server and propagates its height in the new $LLCC$, which results in creating A-DAG, as shown in figure 5.4(h).

Algorithm 5 Reference levels propagation

Action 3: *(i loses all its strong outgoing links)* →
1: **if** $(SN_i = \phi)$ **then**
2: $PI_i := (1, t, i)$; $H_i := ([0, 0], 0, 0, 0, i)$;
3: *broadcast* $CreateDag(PI_i, H_i)$; {*i constructs a new DAG*}
4: **else**
5: **if** $Certain_i = 1$ **then**
6: $Certain_i := 0$; $ERL_i := ([t, t], i, 0)$; $\delta_i := 0$;
7: *broadcast* $Failure(PI_i, H_i)$;
8: **else**
9: *HandleUncertainReverse();*
10: **end if**
11: **end if**

Action 4: *(node i has no strong outgoing links due to link reversal following reception of $Failure(PI_j, H_j)$)* →
1: $Certain_i = 0$;
2: *HandleUncertainReverse();*

Procedure *HandleUncertainReverse()*
1: **if** strong neighbors do not have the same reference level{*Condition 1*} **then**
2: $RL_i := \{RL_k | RLI_k = max_{\succ}\{RLI_j | j \in SN_i, RLI_i\}\}$; $Te_i := t$;
3: $\delta_i := min\{\delta_j | j \in SN_i \text{ and } RL_j = RL_i\} - 1$;
4: *broadcast* $Failure(PI_i, H_i)$;
5: **else if** all strong neighbors have the same reference level with $r = 0$ {*Condition 2*} **then**
6: $ERL_i := ([Tb_j, t], oid_j, 1)$; $\delta_i = 0$;
7: *broadcast* $Failure(PI_i, H_i)$;
8: **else if** all strong neighbors have the same reference level with $r = 1$ and $oid_i \neq i$ {*Condition 3*} **then**
9: $old(lid_i) := lid_i$; $old(Tc_i) := Tc_i$; $PI_i := (1, t, i)$;
10: $H_i := ([0, 0], 0, 0, 0, i)$;
11: ***download*** *the service from* lid_i;
12: *broadcast* $CreateWeakDag(PI_i, H_i, old(lid_i), old(Tc_i))$; {*$i$ constructs a weak DAG*}
13: **else if** all strong neighbors have the same reference level with $r = 1$ and $oid_i = i$ {*Condition 4*} **then**
14: $PI_i = (1, t, i)$; $H_i = ([0, 0], 0, 0, 0, i)$;
15: ***download*** *the service from* lid_i;
16: *broadcast* $CreateDag(PI_i, H_i)$; {*i constructs a new DAG*}
17: **end if**

5.5.6 Self-stabilizing subgraph merging algorithm

Algorithm 6 consists of actions dealing with the propagation of a new server throughout a LLCC. As explained in section 5.5.5, the node predicting network partitioning

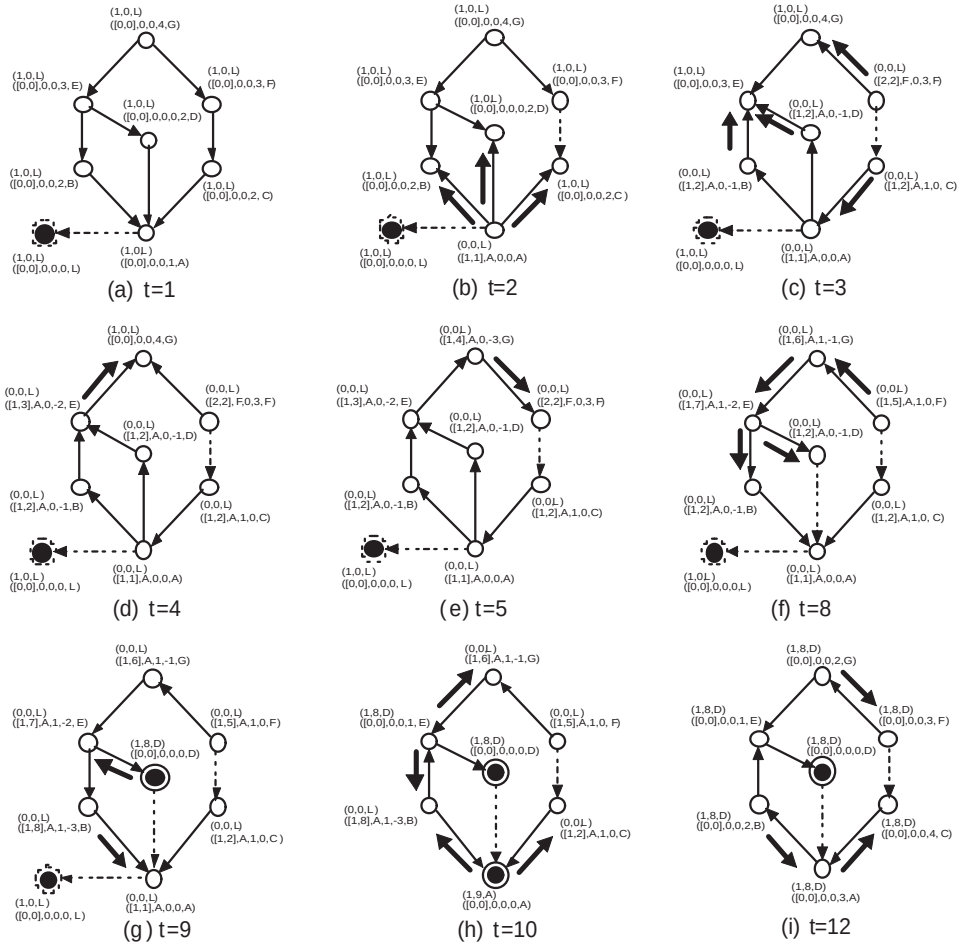


Figure 5.4: An example of executing the self-stabilizing service coverage algorithm

propagates a *CreateDag* message if it is the originator of the reference level or *CreateWeakDag* message otherwise. In *Action 5*, when a node i detects the existence of a new LLCC or receives *CreateDag* message from node j such that $lid_i \neq lid_j$, it will adopt lid_j as a server if the partition index of j has higher priority than that of i, and we write $PI_j \succ PI_i$.

Definition 19. $PI_j \succ PI_i \equiv ((Certain_j > Certain_i) \vee ((Certain_j = Certain_i) \wedge ((Tc_j, lid_j) \succ (Tc_i, lid_i))))$, *and* $(Tc_j, lid_j) \succ (Tc_i, lid_i) \equiv (Tc_j < Tc_i) \vee ((Tc_j = Tc_i) \wedge (lid_j > lid_i))$

By definition 19, node first checks the value of $Certain$ variable. It favorites the

node which is sure that has a directed strong path toward its server node and it is not involved in links reversal actions. Then, it checks the DAG generation time and the server identifier. The condition $PI_j > PI_i$ incurs three cases:

1. If node j is certain and i is uncertain.
2. Nodes i and j both have the same value of certain, and the j's partition is created at a time older than that of i.
3. Nodes i and j both have the same value of certain, and the same value of DAG creation time, and $lid_j > lid_i$

In figure 5.4(g), node A, which predicts network partitioning, sets PI_A to $(1, 9, A)$. Then, it propagates the *CreateDag* message as shown in figure 5.4(h).

If the condition presented in (*HandleUncertainReverse: line 8*) holds true, node i declares itself a server. As i is not the originator of the reference level, it may happen that this declaration is false because both nodes i and the originator of the reference level are still part of the same network partition. In this case, it creates a weak DAG by generating a *CreateWeakDag* message that contains besides its partition index and its height, also the identifier of its previous server and its previous Tc (e.g. node D in figure 5.4(g)). The weak DAG differs from the known DAG in the fact that its propagation can be stopped by an uncertain node. In *Action 6*, when an uncertain node receives the *CreateWeakDag* message from node j, it first checks if the condition $(((old(lid_j) = lid_i \wedge old(Tc_j) = Tc_i) \wedge (certain_i = 1 \vee (certain_i = 0 \wedge r_i = 0))))$ holds true. The first part of the condition means that nodes lid_i and lid_j was part of the same DAG, and the second part means that either node i still has a path toward lid_i or i has not yet reflected its reference level (i.e., the reference level is in the forward phase). This indicates that the creation of a new LLCC was not occurred, and hence the decision of creating a new DAG by lid_j was not correct, and node lid_j is called in this case a fake server. Node i then broadcasts *FakeServer*(lid_j, Tc_j) message. If lid_j is not a fake server, node will adopt lid_j as a server providing that $Pi_j \succ PI_i$. As D-DAG is older than A-DAG, D will be the unique server of the component as shown in figure 5.4(i).

In *Action 7*, each node that find that its server is fake following the reception of a *FakeServer* message from its neighbor j, adopts lid_j as a server and broadcasts in turn the message *FakeServer* message. In this manner, a corrective action will be propagated resulting in deleting the fake server from all the nodes.

Algorithm 6 Merging of two long-lived connected components

Action 5: *((node i detects a new strong neighbor j ∨ receives CreateDag(PI_j,H_j)) ∧ ($lid_j \neq lid_i$)) →*
1: **if** $PI_j \succ PI_i$ **then**
2: **if** ($lid_i = i$) **then**
3: ***Terminate*** *the service;*
4: **end if**
5: $PI_i := PI_j$;
6: $H_i := ([Tb_j, Te_j], oid_j, r_j, \delta_j + 1, i)$;
7: *broadcast* $CreateDag(PI_i, H_i)$;
8: **else**
9: *Stop j-DAG propagation*
10: **end if**

Action 6: *((node i receives CreateWeakDag($PI_j, H_j, old(lid_j), old(Tc_j)$)) ∧ ($lid_j \neq lid_i$)) →*
1: **if** $((old(lid_j) = lid_i \wedge old(Tc_j) = Tc_i) \wedge (certain_j = 1 \vee (certain_i = 0 \wedge r_i = 0)))$ **then**
2: *broadcast FakeServer(lid_j, Tc_j)*;
3: **else if** ($PI_j \succ PI_i$) **then**
4: $PI_i := PI_j$;
5: $H_i := ([Tb_j, Te_j], oid_j, r_j, \delta_j + 1, i)$;
6: *broadcast* $CreateWeakDag(PI_i, H_i, old(lid_j), old(Tc_j))$;
7: **else**
8: *broadcast* $CreateDag(PI_i, H_i)$;
9: **end if**

Action 7: *(node i receives FakeServer(lid_k, Tc_k) from node j)→*
1: **if** ($lid_i = lid_k \wedge Tc_i = Tc_k$) **then**
2: **if** ($lid_i = i$) **then**
3: ***Terminate*** *the service;*
4: **end if**
5: $PI_i := PI_j$;
6: $H_i := ([Tb_j, Te_j], oid_j, r_j, \delta_j + 1, i)$;
7: *Broadcast FakeServer(lid_k, Tc_k)*;
8: **end if**

5.5.7 Time-diagram execution of the self-stabilizing service coverage algorithm

Figure 5.5 shows a time-diagram of executing the self-stabilizing service coverage algorithm related to the example in figure 5.4. The topology changes scenario of this example is the same as the one presented in figure 5.1. The reference level ref_1 that starts at time $t_1 = 1$ blocks the newer reference level ref_2 generated at time $t = 2$, since their reference level intervals belong to the same equivalence class. A weak DAG is generated by node D at time $t = 8$, the network partitioning is predicted by node A at time $t = 9$. The new DAG construction is terminated at time $t = 12$. A-DAG is deleted when it detects the existence of D-DAG with $Tc_D = 8$. The policy that selects the DAG with the oldest value of time creation, helps to block the propagation of any newer DAG created in the same LLCC.

In the partition prediction algorithm, the same scenario needs 17 units of time to recover from three link weaknesses and predict network partitioning, and the DAG

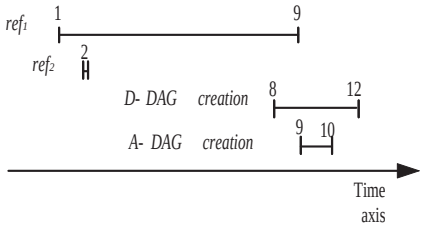


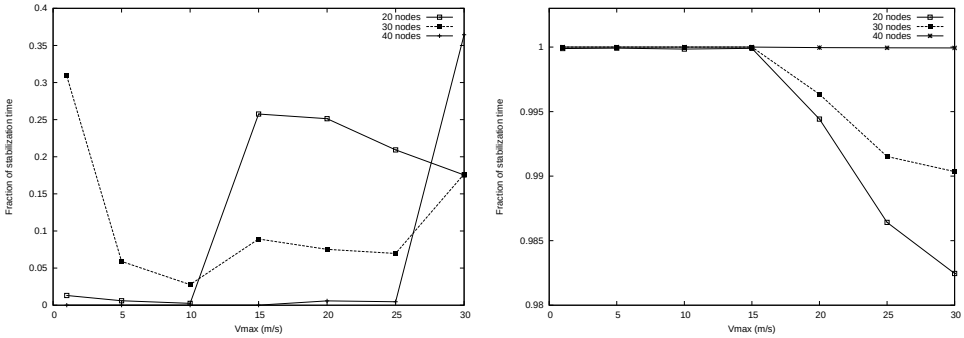
Figure 5.5: Time diagram execution of the self-stabilizing service coverage algorithm

construction will be finished at time $t = 22$. In comparison with the service coverage algorithm, this example shows that the proposed self-stabilizing service coverage algorithm shows better performance in terms of stabilization time and message overhead.

5.6 Simulation Results

In this section, we discuss the performance of self-stabilizing serviced coverage algorithm compared to the service coverage algorithm by using GloMoSim simulator [153]. In our experimental results, each plotted point represents the average of five executions. We plot the 95 % confidence interval on the graphs. Our simulation environment is characterized by an area of 1000m×1000m, with random initial nodes' location, a random waypoint mobility model, in which each mobile node randomly selects a location with a random speed uniformly distributed between 0 and a certain maximum speed V_{max}, then it stays stationary during a pause time of 1 second before moving to a new random location. The transmission power is set to 12 dbm, which is equivalent to a transmission range of $324m$.

We define *Fraction of stabilization time*, which is the fraction of time that a node i has at least a strong path to its server lid_i, and lid_i is the server of the $LLCC$ of node i. Every 1 ms, we monitor the status of the node (i.e., stable or unstable). Thus, if the algorithm can converge to a legitimate state within a time $t < 1ms$, the change of the status is not recorded. The graphs in figure 5.6 show the *Fraction of stabilization time* as a function of V_{max} and for three levels of network densities: low (20 nodes), medium (30 nodes) and high (40 nodes). From the figure, our self-stabilizing service coverage algorithm outperforms the service coverage algorithm since each node can have a path to a correct server over 98% of the time. It is slightly affected by the



(a) service coverage algorithm

(b) self-stabilizing service coverage algorithm

Figure 5.6: Fraction of stabilization time

variation of mobility and node density, but the service coverage algorithm shows disastrous results. The stabilization time of the service coverage algorithm decreases if many nodes are moving toward each other to form a large LLCC, and hence more nodes may trigger their reference levels during the same time period or merge with other LLCCs. It increases when nodes are moving away from each other and the size of the graph becomes small which decreases the number of computations triggered in a LLCC.

5.7 Proof of correctness

Before proving the correctness, we will formally state the definitions of predicates employed by the proof.

- $I_i \equiv (PI_i = null \wedge H_i = null)$
- $P \equiv (\forall i \in C_i : L_i \Rightarrow (\forall j \in C_i, i \neq j : \neg L_j))$, such that: C_i is a long-lived connected component to which i belongs to. P is called the legitimate state.
- $L_i \equiv (lid_i = i \wedge (\forall j \in C_i, j \neq i : lid_j = i \wedge j \textit{ has a stong directed path toward } i))$.
- $dist_i(k)$: is the length of the strong path from i to k.

An illegitimate state is defined to be either: (1) a set of nodes that do not have a strong path to any server, or (2) a set of nodes that have at least a strong path to a

server node but do not have the correct *id* in its *lid* variable.

Assuming that each message transmission takes one time unit, and each node i starts execution in a state satisfying predicate I_i, we prove the the following theorem:

Theorem 1. *After the occurrence of a topological change, the long-lived connected component can within a finite time converge to a state satisfying the predicate P even if further topological changes occur during the convergence time.*

The proof for a single topological change can be found in [100]. We analyze the self-stabilizing service coverage algorithm for concurrent topology changes as follows: for each component C_i, we start observing the algorithm execution when the first topology changes occurs in $C_i(t_s)$. $C_i(t)$ denotes the component to which i belongs to at time t. The observation ends when the algorithm terminates (i.e. none of the actions are enabled at any node in $C_i(t_e)$), and reaches a state satisfying the predicate P. It is obvious that C_i changes as time progresses. We consider the following five cases of concurrent topology changes:

- Concurrent DAG propagations in a network component.
- Concurrent reference level propagations that do not partition the network component in which they occur.
- Concurrent reference level propagations that partition the network component in which they occur.
- Concurrent merging of network components.
- Concurrent merging of network components with concurrent reference level propagations that partition the components in which they occur.

<u>*Case 1:*</u> Let us consider a set of nodes $\alpha_1, \cdots, \alpha_m$, each of which initiates a DAG construction at $T_{\alpha_1}, \cdots, T_{\alpha_m}$ respectively, such that: $(T_{\alpha_1}, \alpha_1) \succ \cdots \succ (T_{\alpha_m}, \alpha_m)$.

Lemma 1. *Concurrent DAG propagations in a network component will collapse in a finite time into a final one that defines a server oriented-DAG.*

Proof. A node i starts DAG propagation by performing *action 1* or *HandleUncertain-Reverse(line 10,16)*. It sets PI_i and H_i to $(1, t_i, i)$ and $([0,0], 0, 0, 0, i)$ respectively.

Then, it broadcasts a *CreateDag* message. Multiple DAG propagations in a network component implies that: $(\exists j, k \in C_i : lid_j = j \wedge lid_k = k)$.

when the PI_j-DAG propagation arrives at node i that is not assigned to any server, i will be annexed to PI_j-DAG. If $PI_j \succ PI_i$, i sets lid_i to lid_j, and hence i has a strong outgoing link toward j. In this case, PI_j will continue its propagation. Otherwise, it will be stopped. α_1-DAG propagation will stop all other DAG propagations triggered after T_{α_1}. At time $(t_{\alpha_1} + D_i)$ all nodes will have the correct server and a strong path oriented toward α_1, where D_i is the diameter of C_i, and thus P holds true.

□

Case 2: Let $\alpha_1, \cdots, \alpha_k$ be a set of nodes belonging to l-DAG, that lose their last outgoing links at $t_1, \cdots, t_k$ respectively, and the network component in which these failures occur remains connected.

Lemma 2. *All reference levels that do not partition the component in which they occur, will eventually be stopped.*

Proof. It is easy to show that if the network component is not partitioned after the generation of concurrent reference levels, it means that there exists at least a border node, which has a direct strong path to the server, and thus P holds true.

□

Case 3: We consider the case where concurrent reference level generations occurring at nodes $\alpha_1, \cdots, \alpha_k$ at $t_1, \cdots, t_k$ respectively, leads to disconnect a component C_i from l-DAG (i.e. creation of new LLCCs).

Lemma 3. *Eventually within a finite time, network partitioning will be predicted, and the disconnected component will define a new server oriented-DAG.*

Proof. A network partitioning is predicted means that there is no border nodes, and a new LLCC is created. So, to predict network partitioning, a reference level must pass through all nodes of C_i in two phases: (i) forward phase (i.e. $r = 0$), and backward phase (i.e. $r = 1$). We assume that there exists a set of nodes $\alpha_p, \cdots, \alpha_q$ such that $(p \geq 1 \wedge q \leq k)$, and their respective RLIs belong to the same equivalence class. We can remark that RLI_{α_p} is the interval with the highest priority in this set. We will now consider two cases:

Case 3(a): If the l-DAG is disconnected at time t_p. α_p generates a reference level. As RLI_{α_p} is the interval with the highest priority, it will terminate any new reference

level generated at time $t_x > t_p$. So, the network partitioning will be predicted at time $(t_p + 2d)$.

Case 3(b): If the l-DAG was not disconnected at time t_p, but the partitioning occurred at a later time t_q. If the propagation of RL_{α_p} is not stopped during its forward phase, the self-stabilizing partition prediction algorithm will take $O(2d)$ time units. By action 4, any new reference level RL_x generated at time $t_x > t_p$ will be stopped and deleted by RL_{α_p}. Otherwise, the propagation of RL_{α_p} is stopped by node m at time $t_p + level_{\alpha_p}(m)$ due to the existing of a strong outgoing link. We consider that a node α_q loses at time $t_q \in [tp, tp + d]$ its last strong outgoing link, which results in network partitioning. So, it generates a new reference level. The reference level RL_{α_q} will terminate any reference level belonging to an older equivalence class, or generated after t_q. When RL_q arrives at node m after at most $d - level_p(m)$ time units, node m compares RLI_{α_p} and RLI_{α_q}. As $Overlaps(RLI_p, RLI_q)$ holds true, node m stops RL_{α_q} propagation and resumes the propagation of RL_{α_p}. RL_{α_p} needs $(d - level_p(m))$ time units to complete the forward phase and another d time units for the backward phase. During this period, any further link weakness will be contained by RL_{α_p}. The network partitioning will be predicted at $(t_q + 3d - 2 \times level_p(m))$. So, the partition prediction time is upper-bounded by $O(3d)$.

By action 3 and action 4, it may happen that during partition prediction algorithm execution, other nodes that are not the originator of the reference level predict partitioning and create a weak DAG. The originator in turn creates its DAG. The node with the highest PI will become the unique server of the new LLCC.

□

Case 4: We consider an infinite merging of components $C_{\alpha_1}, C_{\alpha_2}, \cdots$ with the component C_i. We assume that components merge with C_i at a Poisson rate λ. The diameter of these components follows an arbitrary distribution with a mean $\overline{D}$, and $\overline{X}$ is the average time to send a message over a wireless link.

Lemma 4. *If C_i has the highest PI, the necessary condition so that the algorithm stabilizes is $\overline{X} < \frac{1}{\lambda \times \overline{D}}$.*

Proof. We model the flow of components they merge with C_i as the queuing system $M/G/1$, with a mean inter-arrival time between two consecutive merging of $1/\lambda$, and a mean service time of $(\overline{X} \times \overline{D})$. We define $U(t)$ to be the remaining time required for a DAG propagation to pass through nodes of $C_i(t)$ whose $lid \neq lid_i$. If $U(t) > 0$ (resp,

$U(t) = 0$), the system is said to be busy (resp, idle). We make the observation that the component C_i passes through alternating cycles of busy period and idle period. As C_i has the highest PI, the merging of C_i and any component C_{α_x} triggers the propagation of a *CreateDag* message throughout C_{α_x}. Initially, the component C_i is in a legitimate state ($U(t) = 0$). When C_{α_1} merges with C_i at time t_1, it terminates the idle period and initiates a new busy period. The average remaining time $U(t)$ required so that the *CreateDag* message covers the C_{α_1} component is $(\overline{X} \times \overline{D_1})$, since it would take this period of time if no further merging occurred beyond this instant. As time progresses from t_1, and the *CreateDag* message is propagating, $U(t)$ is reduced at the rate $\overline{X}$ *sec/sec*. At time $t_2 \in [t_1, t_1 + \overline{X} \times \overline{D_1}]$, a new component C_{α_2} merges with the component $\{C_i(t_1), C_{\alpha_1}\}$, and forces $U(t)$ to increase by D_2. $U(t)$ continues to decrease until it reaches the instant t_m, at which it has covered the whole network component $C_i(t_m)$. This terminates the busy period and initiates a new idle period. The idle period is terminated at time t_{m+1} when $C_{\alpha_{m+1}}$ merges with $C_i(t_m)$. It has been proved in [83] that for $\lambda\overline{XD} < 1$, the busy period ends with probability one, and the average length of the busy period (i.e., stabilization time) is $\frac{\overline{XD}}{1-\lambda\overline{XD}}$.

□

Lemma 5. *Given a set of components $C_1, C_2, \cdots, C_k, C_i, C_{i+1}, \cdots$, such that: $PI_1 \succ PI_2 \succ \cdots \succ PI_{k-1} \succ PI_k \succ PI_i \succ PI_{i+1} \succ \cdots$. The concurrent merging of C_i with some or all of these components will terminate within a finite period of time.*

Proof. To calculate the worst case of the stabilization time, we assume that C_i merges with the components in the following order $C_k, C_{k-1}, \cdots, C_1$. When C_i merges with C_k at time t_k , a *CreateDag* message will be propagated throughout C_i, and each node in C_i will set its PI to PI_k. Before the termination of this propagation, the component C_{k-1} merges at time t_{k-1} with the component $C_i(t_k)$. If no further merging was occurred, the component C_i would stabilize within $\overline{X} \times D_i$ time units. The propagation of PI_{k-1} will take no more than $\overline{X}(D_i + D_k)$ before the arrival of C_{k-2}, and so on. As the time interval $[Tc_1, Tc_k]$ is finite, we assume that the number of network components which have higher priority than that of C_i is also finite. Assuming that C_i merges with components which have higher partition indexes than its one. The necessary time interval for a node belonging to $C_i(t_k)$ (i.e. that node does not leave its component) to have the highest partition index is upper-bounded by: $\overline{X}(k \times D_i + (k-1) \times D_k + (k-2) \times D_{k-1} + \cdots + D_2)$. $= \overline{X} \times \overline{D}(k + (k-1) + \cdots + 1)$, which

equals to $(\frac{k(k+1)}{2} \times \overline{X} \times \overline{D})$. After this time interval, the component C_i will have the highest partition index. By lemma 4, in the presence of infinite merges, the algorithm will stabilize if $\overline{X} < \frac{1}{\lambda \overline{D}}$, and it will take on average an additional $(\frac{\overline{XD}}{1-\lambda \overline{XD}})$ time units. Thus, the average stabilization time is upper-bounded by $(\frac{\overline{XD}}{1-\lambda \overline{XD}} + \frac{k(k+1)}{2} \times \overline{X} \times \overline{D})$.

□

Lemma 6. *Given a set of components $C_1, C_2, \cdots, C_k, C_i, C_{i+1}, \cdots$, such that: $PI_1 \succ PI_2 \succ \cdots \succ PI_{k-1} \succ PI_k \succ PI_i \succ PI_{i+1} \succ \cdots$, and the merging of two components occurs when a new strong link is formed between their uncertain nodes. Then, the concurrent merging of C_i with some or all of these components will terminate within a finite period of time.*

Proof. Each network component C_x consists of multiple uncertain subgraphs denoted by U_x and a unique certain subgraph denoted by S_x. We consider that the diameter of an uncertain subgraph U_x is denoted by Δ_x, and the merging of two components C_x and C_y occurs when a new strong link is formed between their respective uncertain subgraphs U_x and U_y. Let us assume the execution of the following scenario: When the subgraph U_x merges with U_y at time t_x and $PI_y > PI_x$, a *CreateDag* message will be propagated in U_x. After traversing Δ_x hops, the *CreateDag* message is stopped by a certain node $p \in C_x$. This latter propagates in turn a *CreateDag* message that traverses U_x and U_y until it reaches a certain node $q \in C_y$. If $PI_q > PI_p$, node q triggers a *lid_q-DAG propagation* through U_y and C_i. From this scenario, the maximum number of steps to complete the merging of such components is: $(2(\Delta_i + \Delta_k) + D_i)$. Let $\overline{\Delta}$ denote the average diameter of uncertain subgraphs. By lemma 5, a component C_i gets the highest priority index after the merging with k components. So, it needs $((4k\overline{\Delta} + \frac{k(k+1)}{2}\overline{D})\overline{X})$ time units. By lemma 4, after C_i has obtained the PI with highest priority, and in the presence of infinite merging, the algorithm will stabilize if $\overline{X} < \frac{1}{f\overline{D}}$, and it will take on average an additional $(\frac{\overline{XD}}{1-\lambda \overline{XD}})$ time units. Thus, the average stabilization time is upper-bounded by $(\frac{\overline{XD}}{1-\lambda \overline{XD}} + (4k\overline{\Delta} + \frac{k(k+1)}{2}\overline{D})\overline{X})$.

□

<u>*Case 5:*</u> We consider the case where concurrent links weakness occurring at nodes $\alpha_1, \cdots, \alpha_q$ at $t_1, \cdots, t_q$ respectively, leads to disconnect a component C_i from l-DAG. During reference level propagations, infinite components $C_{\alpha_1}, C_{\alpha_2}, \cdots$ merge with C_i.

Lemma 7. *If the disconnected component C_i merges only with components of lower partition index, the algorithm will stabilize in worst cases after an average time of*

$(\frac{\overline{XD}}{1-\lambda\overline{XD}}(2+\frac{2}{1-\lambda\overline{DX}}+\frac{1}{(1-\lambda\overline{DX})^2})$. *Otherwise if* $PI_1 \succ PI_2 \succ \cdots \succ PI_{k-1} \succ PI_k \succ PI_i \succ PI_{i+1} \succ \cdots$, *it will take* $(\frac{(k)(k+1)}{2} \times \overline{D} \times \overline{X})$ *additional time units.*

Proof. *Case 5(a):* By lemma 3, a reference level RL_{α_q} which is propagating in C_i needs to traverse D_i hops to complete the forward phase. If during the forward phase, components of lower priority merge with the disconnected component C_i, then the new resulting components will adopt $lid_i = l$ as a server, which is now unreachable. By lemma 5, reference level propagation in a component that has the highest PI while infinite merges are occurring needs on average $\frac{\overline{XD}}{1-\lambda\overline{XD}}$ units of time to cover C_i. The backward phase of RL_{α_q} in lemma 3 will take at most $\frac{\overline{XD}}{1-\lambda\overline{XD}}$ units of time before colliding with RL_{α_p}. At this time, RL_{α_q} will be stopped and RL_{α_p} resumes its forward propagation phase. During this phase, components with low priority continue to merge with C_i. The average time required to cover C_i is calculated as follows: The diameter of component C_i is increased at the rate of $\lambda\overline{D}$ hops/sec. At the end of RL_{α_q}'s backward phase, the diameter of C_i is upper-bounded by $D_0 = (\frac{\overline{XD}}{1-\lambda\overline{XD}}\lambda\overline{D})$. Therefore, RL_{α_p}'s forward phase needs $T_0 = D_0\overline{X}$ time units to traverse this new diameter. During T_0, C_i is increased by $\lambda\overline{D}T_0$ hops, which means an additional $T_1 = \lambda\overline{DX}T_0$ units of time are needed so that RL_{α_p} completes its forward phase. In the same way, during T_1, C_i's diameter is increased by $\lambda\overline{D}T_1$ hops and so on. Formally: $\sum_{n=0}^{\infty} T_n = (\frac{\overline{XD}}{1-\lambda\overline{XD}})\sum_{n=0}^{\infty}(\lambda\overline{DX})$. Providing that $\lambda\overline{DX} < 1, \sum_{n=0}^{\infty}(\lambda\overline{DX}) = \frac{1}{1-\lambda\overline{DX}} < \infty$. Thus, the forward phase duration of RL_{α_p} is $\sum_{n=0}^{\infty} T_n = \frac{\overline{XD}}{(1-\lambda\overline{DX})^2}$. The backward phase of RL_{α_p} will take the same time as the forward one. The time period to predict network partitioning is upper-bounded by $(\frac{\overline{XD}}{1-\lambda\overline{XD}}(2+\frac{2}{1-\lambda\overline{DX}}))$. At the end of RL_{α_p}'s backward phase, the diameter of C_i becomes $D_0^{new} = \lambda\overline{D}\sum_{n=0}^{\infty}(T_n)$. So, α_p-DAG propagation needs $T_0^{new} = D_0^{new}\overline{X}$ time units to traverse D_0^{new} hops. During T_0^{new}, C_i increases by $\lambda\overline{D}T_0^{new}$. Using the same method as in RL_{α_p} forward phase, the α_p-DAG propagation in C_i component will terminate after $\sum_{n=0}^{\infty} T_n^{new} = \frac{\overline{XD}}{(1-\lambda\overline{DX})^3}$. Therefore, the average stabilization time is upper-bounded by: $(\frac{\overline{XD}}{1-\lambda\overline{XD}})(2+\frac{2}{1-\lambda\overline{DX}}+\frac{1}{(1-\lambda\overline{DX})^2})$ time units.

Case 5(b): C_i merges with k components of higher partition index. By lemma 5,

the algorithm needs an additional $(\frac{k(k+1)}{2} \times \overline{DX})$ units of time to stabilize. In this case the average stabilization time is upper-bounded by:

$$(\frac{\overline{XD}}{1-\lambda\overline{XD}})(2+\frac{2}{1-\lambda\overline{DX}}+\frac{1}{(1-\lambda\overline{DX})^2})+(\frac{k(k+1)}{2}\times\overline{DX})$$

□

The previous seven lemmas prove **Theorem 1**.

5.8 Conclusion

We have shown that the execution time of the service coverage algorithm is unbounded in the presence of concurrent topological changes, and hence it is not possible to give a lower bound for χ_{th} to trigger the partition prediction algorithm.

To deal with this issue, we have proposed a self-stabilizing coverage algorithm that can converge to a legitimate state even in the presence of topological changes during convergence time. By defining concurrent and disjoint computations and their corresponding intervals, an older reference level encompasses any new one belonging to its equivalence class. In the same way, an older DAG propagation encompasses newer ones. Simulation results show that the self-stabilizing algorithm experiences very high service availability compared to the service coverage algorithm. It can ensure that each node has a strong path to the server of its network component over 98% of the time. We have provided a novel observation about self-stabilization by defining the stabilization time by the time period that begins when a component C that contains a node i enters an illegitimate state, and terminates when this particular node will be again part of a component in a legitimate state. Using a Markov chain model, we have shown that the average stabilization time is bounded.

Chapter 6

Location-based data replication protocols

6.1 Introduction

In the literature, some of the replication protocols [142, 26, 65] that have been proposed for ad-hoc networks try to improve data availability by predicting the time at which network partitioning might occur and replicate data items or services beforehand. However, these solutions are not scalable because: (1) the partition prediction algorithms are themselves not scalable, and (2) nodes in some cases have to invoke a topology-based routing protocol to send the update and query messages. Another category called the location-based replication protocols [135, 120, 125] can scale to large number of nodes, since they use the position-based routing protocol to route their control packets. However, this category suffers from the fact that it does not predict network partitioning, which results in low data availability.

In this chapter, we propose three location-based replication protocols: FDRS, HDRS, and a location service FSLS [38], which is a succession of our location service (*ELS*) [42]. The proposed protocols show good results and offer trade-off between availability and scalability metrics.

6.2 Flat-based some-for-some location service (FSLS)

Ad hoc networks characteristics impose different challenges on designing location services. First, dynamic topology leads that there is no static relation between a node and its location. To distribute location information to a set of servers, the location service utilizes a routing protocol, but the routing protocol requires the location information of these servers in first place, which results in a functional deadlock. Second, the location service must incur low overhead in order that the servers do not quickly drain out their batteries. Third, network partitioning would make the location servers unreachable to some nodes, and hence it considerably decreases the availability of location information. In other words, the location service that would be deployed in the ad hoc networks needs to be scalable and provide high location availability. A location service is said to be scalable if it has the ability to adapt to increased demands in terms of network size and node speed without the need for significant additional overhead. The location availability is defined as the probability to access location information when needed under dynamic environment caused by node mobility and node failures. Recently, many location services for ad hoc networks have been proposed [145, 28, 91, 147, 151, 126, 32]. Many of which have only focused on designing scalable solutions without paying too much attention to their location availability.

6.2.1 Design Considerations

In the literature, two metrics are used to assess the efficiency of any location service: the scalability in terms of communication cost and location availability. The communication cost is defined to be the average number of messages a node generates per unit time to perform an update or a query operation, and the location availability.

Yu et al. [152] have analyzed the scalability of SLURP [145], SLALoM [28], GLS [91], DLM [147], and HIGH-GRADE [151]. The analysis has shown that the communication cost of HIGH-GRADE scales linearly with the node speed v and logarithmically with the number of nodes in the network n. HGRID [117] also has an asymptotic communication cost of $O(v \log(n))$. They outperform the other location services. SLALoM has an asymptotic communication cost of $O(v\sqrt[3]{n})$, versus $O(v\sqrt{n})$ for SLURP and and the quorum-based location service (CRLS) [130].

Figure 6.1 presents the above location services along two dimensions: scalability

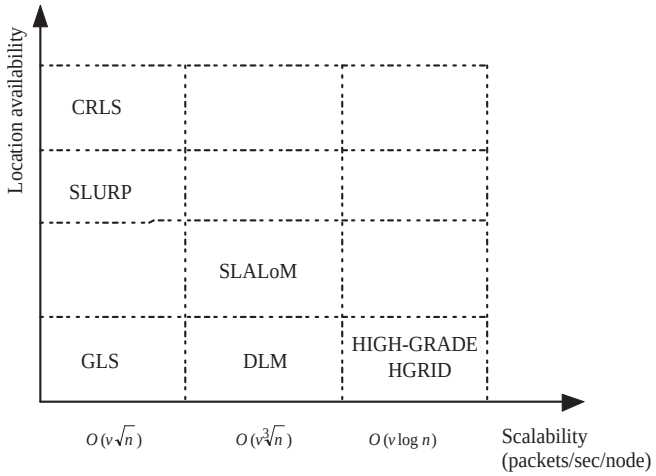


Figure 6.1: Location services along two metrics

and location availability. Figure 3.6(d) shows that HIGH-GRADE imposes on querying nodes to contact a chain of servers in a certain order so as to retrieve the queried node's exact location. If one of these servers is unreachable or the area, which is supposed to contain a location server is empty, the query operation cannot be performed. From the figure, we can intuitively conclude that the location availability of multi-level location services is low. This intuition is proven in Section 6.2.6. The same thing can be said about GLS and DLM as they are also multi-level location services. SLALoM has better location availability than HIGH-GRADE since it needs to contact at most two location servers. As query packets in SLURP do not need to contact many servers in a certain order or to traverse specific areas to reach their target regions, its location availability is higher than that of SLALoM. However, it still suffers from unsuccessful location query due to empty regions. CRLS outperform the other location services in terms of location availability because the intersection of row and column is guaranteed. As both metrics are important for ad-hoc networks, we propose a location service that balance the tradeoffs between scalability and availability.

6.2.2 System Framework

The architecture of the proposed system framework is shown in Figure 6.2. It consists of four layers: the application layer, the middleware layer, the routing layer and the MAC layer. As part of the framework, the routing and the MAC layers provide an estimation of the residual node lifetime and the available bandwidth respectively.

When a location-aware application such as the tour guide wants to obtain the position of a certain node D, it contacts the location service. This latter will check if such a position is available in its own location table. If so, it will directly respond to the application. Otherwise, it will demand the position-based routing protocol to find at least one of the location servers of D. The location servers should respond with the current location of D. On the other hand, when a node S wants to communicate with D, it first contacts the routing protocol, which in turn contacts the location service. The location servers of D should respond with an approximative location information, i.e., the geographic area where D is located. In the cross-layer information, the routing layer estimates the residual battery lifetime by recording the different packets that pass through it. It can be used to determine the time at which the location information table will be replicated onto another node, in order to maintain location information availability for a long time period. The available bandwidth estimation provided by the MAC layer is used to derive an upper-bound on the replication time.

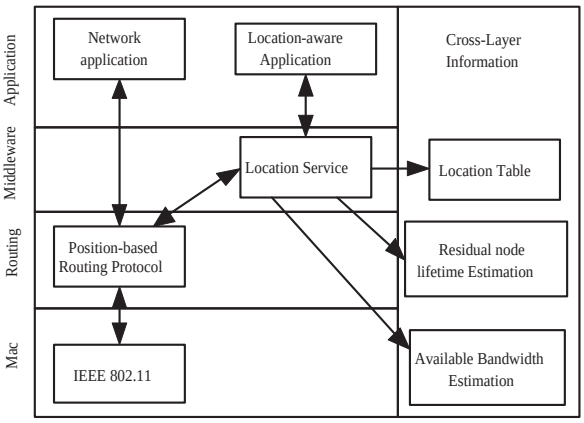


Figure 6.2: System framework

6.2.3 Definitions

We propose a new construction that maps each node identifier to a fixed set called *a hash-based set*. A *hash-based set* for a node A is obtained by applying a known hash mapping algorithm that maps each node A to a set H_A.

We define the bi-set B_A as a couple (U_A, Q_A), such that $U_A = H_A$, and $U_q \subseteq U_A$. U_A and Q_A are called the *hash-based update set (HUS)* and the *hash-based query set (HQS)* of node A respectively.

The hash-based sets system $\mathcal{S}$ is a set of bi-sets, i.e., $\mathcal{S} = \{B_1, \cdots, B_m\}$. We consider that $m \leq n$ because two nodes may map to the same hash-based set.

6.2.4 Area partitioning and location server selection

The first design choice we adopts is structuring the location servers as a flat-based approach. The area covered by the ad hoc network is partitioned into G square zones of equal-sizes. All of these zones have well-known identifiers (IDs) distributed over the range $[0, \cdots, G-1]$. We assume that there exists a static function f that maps a node's ID into a specific zone. Formally, f(node ID) $\rightarrow$ zone ID. Figure 6.3 depicts the partitioning scheme, in which the respective zones' identifier are shown in the upper-left corner of each zone, and the location servers are colored gray. Each node is assigned a hash-based update set (*HUS*) consisting of α zones. α is a system parameter upper-bounded by G. A well-known hash algorithm $\mathcal{H}(A, G, \alpha)$ constructs a *HUS* for node A (Algorithm 7), where U_A represents the set of zones' identifier. An example of a function f is: $f(ID) = (ID\%G)$. Nodes that return the same value of $f()$ are assigned to the same *HUS*. In figure 6.3, $f(A) = 0$, $G = 20$, and $\alpha = 3$. So, U_A is $\{0, 6, 12\}$.

Algorithm 7 The hash algorithm $\mathcal{H}(A, G, \alpha)$

1: $U_A = \emptyset$;
2: $k := 0$;
3: **for** $j = 1$ to α **do**
4: $\quad U_A = U_A \cup \{f(id(A) + k)\}$;
5: $\quad k := k + [\frac{G}{\alpha}]$;
6: **end for**
7: ***return*** U_A;

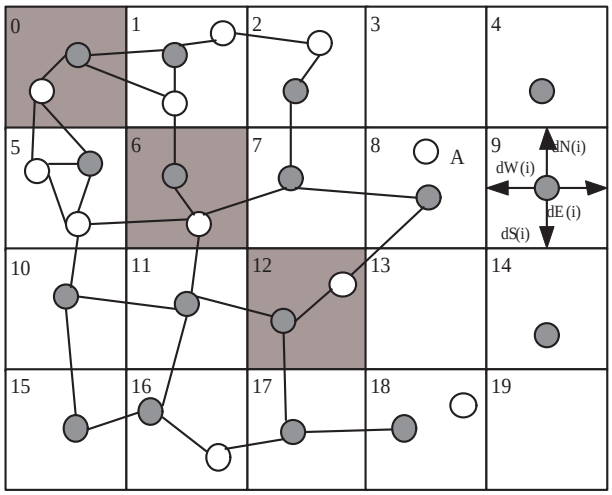


Figure 6.3: Area partitioned according to the flat-based approach

Unlike other flat-based location services, the hash algorithm maps a node identifier to a set of home zones. Moreover, only a unique node from each zone is selected as a location server. In real mobility patterns, node density may not be uniform across the network, which results in multiple empty zones. In this scheme, a query operation fails only if the querying node cannot access any of its location servers.

We propose the *stability of a node* as a metric for the location server selection criterion. The metric predicts the time period during which the node will remain in its zone. It is based on the residual battery power and the current positions of the node. A node disappears from its zone either: (i) if it moves out of its current zone, or (ii) or it drains out of its energy power. Each node i estimates its stability as follows: First, it calculates the rate of its battery power depletion R_i for every time period $\triangle T$, then it estimates the residual time before it runs out of energy power, as shown in equation 6.1, such that E_i denotes i's residual battery power. Second, it calculates the remaining time before it moves out of its current zone, $Tmob_i$ (equation 6.2). Where $dN(i)$, $dS(i)$, $dE(i)$, and $dw(i)$ are the distances that separate node i from the north, south, east, west sides of its current zone, and $Vmax_i$ is the node i's maximum velocity. The respective distances are depicted in figure 6.3.

$$Tpow_i = \frac{E_i}{\max(R_i)}(s) \tag{6.1}$$

$$Tmob_i = \frac{\min\left(dN(i), dS(i), dE(i), dW(i)\right)}{Vmax_i}(s) \tag{6.2}$$

Equation 6.2 derives a conservative estimate on $Tmob_i$. It assumes that the node is moving toward the nearest side. Finally, the stability of node, denoted by i ξ_i is given by the following equation.

$$\xi_i = \min\left(Tpow_i, Tmob_i\right) \tag{6.3}$$

Initially, the node with the highest value of stability will be selected as the location server of its zone. Ties are broken by comparing node identifiers.

The Combination of the hash-based sets construction, the flat-based approach, and the location server selection procedure leads to the creation of G servers. We assume that $G < n$. Each server node stores the location information of $(\frac{\alpha \times n}{G})$ nodes. As $(\frac{\alpha}{G}) < 1$, the proposed location service (FSLS) can be classified as a some-for-some approach.

6.2.5 Service operations

Nodes perform two types of operations: *update* and *query*. Each operation has its corresponding response, *ack* for update and *reply* for query. Each location server holds a location table, which records for each stored node the following fields: (1) its id, (2) the zone id where that node is in, and (3) its tiemstamp (i.e. the latest update time known by the location server). As we can see, FSLS adopts a two-level grained strategy. We assume that the position-based routing protocols are much more invoked than the location-ware applications, and they aim to deliver data packets to the destination nodes, and not to know the exact location of those nodes. The exact location of node is only known by the node itself and its neighbors.

Location update

When a node i moves out of its current zone and into a new one Z_n, it sends an *update* message toward the center of each zone $\in (U_i \cup Z_n)$. The update packet contains the following information:
$<$ *src id*, *seq*, *src zone*, *target zone*, *new zone*, *new timestamp* $>$. The pair (*src id*, *seq*) denotes the source node and the sequence number of the packet. It uniquely

identifies the update packet. *src zone* is the id of the source node's zone and *target zone* is the server node's zone. *new zone* is the node's new zone. The update packet also includes the new timestamp which is obtained by increasing the current timestamp by one.

When the packet reaches a node in the target zone, two cases can occur. If that node has a cached route to the location server, the packet is immediately routed toward the server. Otherwise, a route discovery packet is broadcasted in the zone to find a route to the location server. Upon receiving the update packet, the location server sends back an ack packet to the source node.

Location query

According to users and applications requirements, query packets can be sent with two levels of accuracy: high, and low:

- *High accuracy:* The exact position of the target node is needed by the location-aware applications.
- *Low accuracy:* It is required by the position-based routing protocols that aims to know the zone id where the target node resides in and not to know its exact location.

The major disadvantage of the flat-based location services is that a query packet sent by a node may need to traverse long distances to retrieve the location of nearby destinations, therefore increasing query latency. To deal with this issue, we propose to limit the distance that the query packet traverses. A source node B wishing to obtain the position of a node A, firstly sends a query packet to the location server of the zone where it resides, denoted by L_B (arrow (1) in figure 6.4). If L_B holds the location information of D, then S will obtain D's location within a time proportional to the side length of the zone. If node B receives no response, it will execute Algorithm 8 to obtain the Q_A set. Q_A is first defined to be the set of zones that belong to U_A such that the physical distance between node B and the center of each zone is bounded by d_{th} (arrow (2) in figure 6.4). If the resulting Q_A is empty, Q_A will be set U_A. Then, B will send a query packet toward the center of each zone $\in (Q_A \cup Z_c)$.

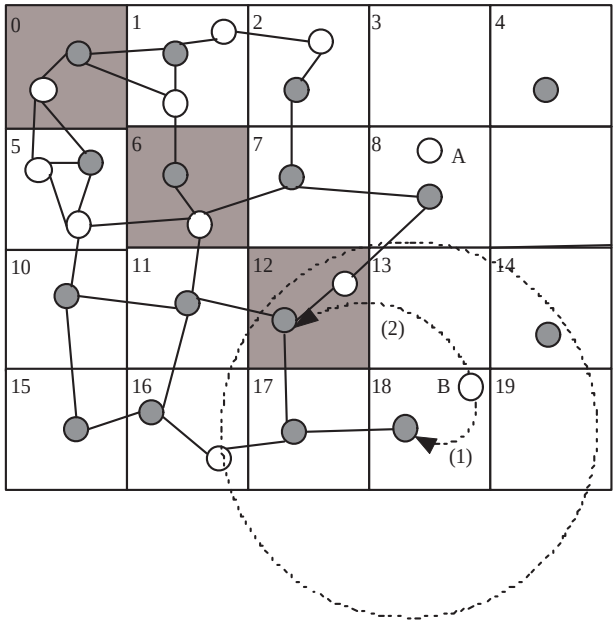


Figure 6.4: An example of executing a query operation

Unreachable zones

If a node i fails to contact the server of a certain zone, it considers that the zone is unreachable. A zone is called unreachable if either it is empty or the location server in that zone is unreachable due to network partitioning. To avoid unnecessary message overhead, each node keeps a set, called the *Unreachable zone list (UZL)*. This latter includes the zones' ID which are unreachable. When a given zone is declared as unreachable, it will be added to the *Unreachable zone list* for a period of time T. During that period, update and query packets are sent to $(U_i - UZL_i)$ and $(Q_i - UZL_i)$ respectively. After the expiration of T, the zone in question will be removed from the list.

Handling mobility and failure of location servers

When a location server becomes no longer available, queries that arrive between the time that the server is unavailable and the next updates from nodes whose locations are stored at the server will fail. To address this issue, the location server node that is about to cross the boundary of its zone or it is about to run out of battery power, has

Algorithm 8 Construction of Q_A set

1: $Q_A := \emptyset$
2: $U_A := \mathcal{H}(A, G, \alpha)$;
3: **for** *all* Z *in* U_A **do**
4: **if** $(distance(B, Z) < d_{th})$ **then**
5: $Q_A := Q_A \cup \{Z\}$;
6: **end if**
7: **end for**
8: **if** $(Q_A = \emptyset)$ **then**
9: $Q_A := U_A$
10: **end if**
11: ***return*** Q_A;

to replicate the location information table onto a new node before it leaves the zone or it dies. The service replication is triggered when the following constraint is verified: $(\xi_i - \Delta T) <$*time of replication process*. Let us consider that D denotes the size of location information table to be replicated and transmitted over an end−to−end wireless connection with an available bandwidth of Bw bits/s. The replication process needs $(\frac{D}{Bw})$ seconds in order to be achieved. The old server i and the new server j must fulfill the following requirements: (1) $(\xi_i(t) - \Delta T) < (\frac{D}{Bw})$ and (2) $\xi_j(t)$ is the maximum among the nodes of its zone.

The problem that can occur when tracking the residual lifetime of a node in its zone, is its disappearance during the interval Δt. No warning is generated until the next period. By that time, the server may be already disappeared or not enough time is left to carry out the replication process. To efficiently track the node in its zone, the following property must hold true $\Delta t < \max(\frac{1}{Vmax_i}, \frac{1}{\max(R_i)})$. This technique ensure any-time service availability as long as the zone is not empty.

It is possible all the α zones of a node are empty, and thus the update and the query operations fail because the querying node node does not receive any response. To deal with this problem, and in order to reduce the effect of such cases on data availability, we propose a known rescue hash mapping algorithm that maps each node A to a new set denoted by H_A^r.

6.2.6 Analysis

In this section, we use a common theoretical framework to compare the scalability and the availability of FSLS, HIGH-GRADE, GLS, DLM, SLURP, SLALoM, and CRLS under: (1) uniform and localized[1] traffic patterns. The notations used in this section are summarized in Table 6.1. We define four metrics: location update cost, location quer cost, storage cost and availability degree, which are formally defined as follows:

- *Location update cost (c_u):* The average number of one-hop transmissions each node needs to perform in a second to maintain fresh location information on the location servers.
- *Location query cost (c_q):* The average number of one-hop transmissions due to location queries each node needs to perform in a second.
- *Storage cost (c_s):*): The number of location records all the location servers store.
- *Location availability (Ad):* The probability that a query operation acquires the location information of the queried node.

Table 6.1: Notations

p_m	prob. that distance between 2 nodes is $\leq r$
p_a	prob. that a node is alive
C	battery power capacity
g	prob. of establishing a link between 2 nodes
P_i	prob. of establishing a path of i hops between two nodes
m_i	average number of nodes that are located in the same level-i square as A but not in the same level-j such that $j < i$
ρ_i	level-i square boundary crossing rate
d_i	distance between two random points in a level-i square
n_i	number of hops between two random points in a level-i square
z	average progress of each forwarding step
c_1	constant of random distance within a square
r_0	the minimum transmission range for which the network is connected
R	side length of a level-0 square

6.2.7 Scalability analysis

Woo and Singh [145] have proposed a theoretical framework to analyze the scalability of a location management. Under this framework, n nodes are randomly distributed

[1]localized means that queries are from nearby nodes

in a region of area A. Nodes select a random direction to move in, chosen uniformly between $[0, 2\pi]$, and a random distance for which the move. each node also selects its speed v, chosen uniformly between $[v - c, v + c]$. The average degree of a node (i.e., node density) is constant, the area A must grow with n. The area of each zone is a. Thus, The ad hoc network region is divided into $G = (\frac{A}{a})$ zones. The main observations from [145] are the following:

1. The cost of broadcasting in a square by a node, b, is proportional to the number of transmissions needed to cover the said square. Thus, $b = (\frac{a}{r^2})$, where r is the transmission radius of a node.

2. The distance a node has to cover to cross an unit square is proportional to the side of that unit square. Thus, the number of zones a node crosses per second, ρ, is proportional to $\frac{v}{\sqrt{a}}$.

3. Given two nodes separated by distance d, the number of hops needed to send a packet from one node to the other is given by $u = \frac{d}{z}$, where z is the average forward progress made toward a destination in the course of one transmission. z depends on r_t and the average degree of a node.

Location update cost

When a node moves out of its current zone and into a new one, it generates an update message that is sent to $\overline{X} = (p \times \alpha)$ zones. In addition, the update packet is sent to the server of its new zone. When the update packet reaches a node of one of those zones. If that node has not a cached route to the location server, it broadcasts a route discovery packet to all nodes within the zone to establish a route to the server. The cost of updating one server is $(b + u)$. The location update cost can be written as:

$$c_u = O(\rho(\alpha(b + u) + \sqrt{a})) \text{ packets/sec/node} \tag{6.4}$$

The average distance between two random points in a square of area A (i.e., u) is proportional to $\sqrt{A}$ [45]. The location servers incur additional overhead, since before they move into a new zone, they have to elect a new server. The election procedure requires $O(a)$ broadcasts. The total location update cost (c_{Tu}) is:

$$c_{Tu} = O(\rho[\alpha(\frac{A}{a}(u + 2b) + (n - \frac{A}{a})(u + b)) + n\sqrt{a}]) \text{ packets/sec} \tag{6.5}$$

Substituting ρ, u and b, we have

$$c_{Tu} = O(\frac{\alpha v}{\sqrt{a}}[\frac{A}{a}(\sqrt{A} + 2a) + (n - \frac{A}{a})(\sqrt{A} + a)] + v \times n) \text{ packets/sec}$$

Recall that A is proportional to n, therefore, we get

$$c_{Tu} = O((\alpha v)(\frac{n}{\sqrt{a}} + \frac{n^{\frac{3}{2}}}{\sqrt{a}} + n\sqrt{a}) + v \times n) \text{ packets/sec}$$

We hold α as constant, therefore, we get

$$c_{Tu} = O(v(\frac{n}{\sqrt{a}} + \frac{n^{\frac{3}{2}}}{\sqrt{a}} + n(1 + \sqrt{a}))) \text{ packets/sec}$$

Minimizing c_{Tu} with respect to a, we have $a = O(\sqrt{n})$ and thus the total location update cost is

$$c_{Tu} = O(v(n^{\frac{5}{4}} + n + n^{\frac{3}{4}})) = O(vn^{\frac{5}{4}}) \text{ packets/sec}$$

Therefore, the location update cost is $O(v\sqrt[4]{n})$ packets/sec/node

Location query cost

When a node wishes to find the location of a specific target node, it sends a query packet to β zones. When a node of the target zones receives the packet, and it has not cache to its location server, it broadcasts a route discovery packet to all nodes within the zone. The cost of performing this query is $(u + b)$. If the query packet is of high accuracy, the location servers need to contact the queried node itself. We assume that queries with high and low accuracy are issued by each node at a Poisson rate of λ_H query/sec and λ_L query/sec respectively. The query cost (c_q) is

$$c_q = O(\beta(\lambda_L(u + b) + 2\lambda_H(u + b))) \text{ packets/sec/node} \tag{6.6}$$

c_q can be written as

$$c_q = O(\beta(\lambda_L + 2\lambda_H)(\sqrt{n} + a)) \text{ packets/sec/node}$$

λ_H, and λ_L are constant, therefore, eliminating terms and substituting a with $\sqrt{n}$, we get

$$c_q = (\beta\sqrt{n}) = O(\sqrt{n}) \text{ packets/sec/node}$$

If the querying node and the target node both reside in the same zone, The query packet traverses a distance of $O(\sqrt{a})$. So, the query cost for localized traffic is $O(\sqrt[4]{n})$ packets/sec/node.

Storage cost

Each location server stores $(\frac{\alpha \times n}{G})$ records, and there exist G location servers. As α is constant, the storage cost is proportional to n.

6.2.8 Availability analysis

The probability density function (pdf) of the distance S between two nodes with uniformly distributed node is given by [18]:

$$f_S(s) = \frac{\widehat{s}}{9\pi}[18\pi - 36\arcsin(\frac{\widehat{s}}{2}) - 9\widehat{s}\sqrt{4-\widehat{s}^2}]$$

where $\widehat{S} = \frac{S}{a}$, and $a = \sqrt{\frac{A}{\pi}}$.

We assume that all nodes have the same transmission radio range. Two nodes establish a link if they are located within distance of r of each other. The probability that the distance between two nodes is less than or equal to r is given by [47]:

$$p_m = P(S \leq r) = \int_0^r f_S(s)ds$$

We use a birth-death process for modeling changes in the amount of residual battery power at a mobile node. We denote with Q_k the state of the system when the residual battery power R is k $Joules$. From the state Q_k, birth-death process may transit only in state Q_{k+1} and state Q_{k-1}, or remain in the state Q_k during time interval Δt. R is increased with the rate of λ $Joules/s$, and it is decreased with the rate of μ $Joules/s$, such that $\lambda < \mu$. The battery power capacity is denoted by C.

Using Kendall's notation [17], we describe the system by using $M/M/1/C$ queuing system. By assuming equilibrium in the system, we may apply global balance equations. We denote by Π_i the steady state probability that the system is at state

E_i.

$$\Pi_i = \frac{1-\frac{\lambda}{\mu}}{1-(\frac{\lambda}{\mu})^{C+1}}(\frac{\lambda}{\mu})^i \quad 0 \leq i \leq C$$

The probability that a node is alive is given by:

$$p_a = \sum_{i=1}^{C} \Pi_i = 1 - \Pi_0 = \frac{\frac{\lambda}{\mu} - (\frac{\lambda}{\mu})^{C+1}}{1 - (\frac{\lambda}{\mu})^{C+1}}$$

A link (X, Y) is formed if both endpoints X and Y are alive and they are within the transmission range of each other. The probability of establishing a link is obtained as : $g = p_m \times p_a^2$.

If the two nodes are not neighbors, they can indirectly communicate with each other by using the intermediate nodes. Two arbitrary nodes in the network with n nodes connected by i hops through $(i-1)$ intermediate nodes. There are $(n-2)$ ways that any arbitrary (source, destination) pair can set up a path of two hops, $(n-2)(n-3)$ ways to set up a path of three hops, and $(n-2)(n-3)\cdots(n-i)$ ways to set up a path of i hops. Thus, the probability of establishing a path of i hops is given by:

$$P_i = K_i g^i; 1 \leq i \leq n-1$$

where

$$g^i = p_a(p_a p_m)^i$$

$$K_1 = 1$$

$$K_i = (n-i)K_{i-1}; 2 \leq i \leq n-1$$

The network is connected if and only if there is a path between each pair of nodes. The probability that the network is connected is then obtained as $Pc = \sum_{i=1}^{n-1} P_i$.

For a given, n, there is r_0 such that: $\sum_{i=1}^{n-1} K_i g^i(r_0) = 1$, where r_0 is the minimum transmission range for which the network is connected. For any $r > r_0$, there is a

$n' < n-1$ such that: $\sum_{i=1}^{n'-1} P_i < 1$, and $\sum_{i=1}^{n'} P_i \geq 1$. Therefore, P_i is given by

$$P_i = \begin{cases} K_i g^i(r) & \text{if} \quad 1 \leq i \leq n'-1 \\ 1 - \sum_{i=1}^{n'} K_i g^i(r) & \text{if} \quad i = n' \\ 0 & \text{if} \quad n' < i \leq n-1 \end{cases}$$

In [18], it is shown that the p_m is increasing function of the transmission radio range r. It is easy to see that P_i is dependent on the number of nodes n, and the transmission range.

Here we can distinguish two cases: $\{P_i\}$ is a decreasing or an increasing sequence.

We will prove by induction that if $(n-2)g < 1$, then $\{P_i\}$ is a decreasing sequence for every $i \in [2, n'-1]$.

Proof. Base case: $i = 2$, we get $P_2 = (n-2)P_1 < 1$, and hence $P_2 < P_1$.

Inductive Hypothesis: Let us assume that $\{P_i\}$ is a decreasing sequence for every l such that: $2 \leq l \leq i$. We shall now prove that $\{P_l\}$ is a deceasing sequence for $l = i+1$.

P_i can be written as follows: $P_i = (n-i)gP_{i-1}$. As $P_i < P_{i-1}$, then $(n-i)g < 1$. We have also: $P_{i+1} = (n-(i+1))gP_i$, and as $(n-(i+1))g < (n-i)g < 1$, then $P_{i+1} < P_i$. Therefore, $\{P_i\}$ is a deceasing sequence for every l such that: $2 \leq l \leq n'-1$.

□

We can easily show that if $(n-i)g > 1$ for every $i \in [2, n'-1]$, then $\{P_i\}$ is an increasing sequence.

In hierarchical-based location services (HIGH-GRADE, DLM, and GLS), the entire network area is called a level H square. The level H square is divided into four quadrants, called the level-$(H-1)$ squares, each of which is further divided into four quadrants as well, so and so on forth, until the entire region is divided into 4^H level-0 square. The side length of a level-0 square is denoted by $R = \frac{\sqrt{A}}{2^H}$.

The average number of nodes m_i (see Figure 6.5) that are located in the same level-i square as A but not in the same level-j such that $j < i$, can be easily obtained as follows:

$$m_i = \begin{cases} \frac{3n}{4^{H-i+1}} & \text{if} \quad 1 \leq i \leq H \\ \frac{n}{4^H} - 1 & \text{if} \quad i = 0 \end{cases}$$

We can obtain that: $\sum_{i=0}^{H} m_i = n - 1$

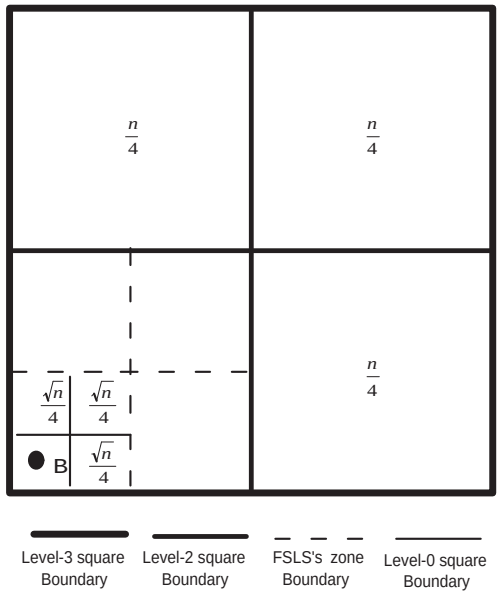


Figure 6.5: Node distribution in the network

HIGH-GRADE

In HIGH-GRADE, If nodes A and B are co-located in the same level-i square, the query is forwarded sequentially from $pLSP_0, pLSP_1, \cdots$ up to $pLSP_i$ (LSP_i) then to LSP_{i-1} until LSP_0. In this case, the probability Ad_i^{HG} that a node B obtains A's exact location information is:

$$\begin{aligned} Ad_i^{HG} &= \prod_{j=0}^{j=i} P_{\phi 2^j} \times \prod_{j=i}^{j=1} P_{\phi 2^j} \\ &= P_\phi \prod_{j=1}^{j=i} P_{\phi 2^j}^2 \end{aligned}$$

GLS

In GLS, each mobile node may designate nodes in each sibling region with IDs closest to its own ID to serve as its location servers. When a node B wants to find the location of A, it sends a query packet toward a node whose ID is the least greater

than or equal to the A's ID within the order-1 square. The query packet is then re-sent to the node in the next order square in the grid hierarchy, whose ID is the least greater than or equal to the A's ID within the order-2 square, and so on.

If i is the level of the minimum common square A and B are co-located, the probability Ad_i^{GLS} that a node B obtains A's exact location information.

$$Ad_i^{GLS} = \prod_{j=0}^{j=i} P_{\phi 2^j}$$

DLM

DLM partitions the entire network like GLS. There are $H+1$ levels of squares. The location servers are distributed uniformly across the network, one server in every level-k square. If the server is located in the same level-k square as A, the complete location information is stored. When a node B wants to find the location of A, and the partial address policy is used, a query packet is sent toward the location server of its level-k square. If the complete address of A is found, the query is complete. Otherwise, the server of A indicates which level-j square A is in such that $j > k$.

If i is the level of the minimum common square A and B are co-located, the probability Ad_i^{DLM} that a node B obtains A's exact location information is:

$$Ad_i^{DLM} = P_{\phi 2^k} \prod_{j=k}^{j=H} P_{\phi 2^j}$$

SLURP

In SLURP, the query packet is forwarded to A's home square. The probability to retrieve A's location information is $Ad^{SLURP} = P_{\phi 2^H}$.

SLALoM

The probability that a node B obtains A's exact location such that A's home square is in the level-1 square B is in, is given by $Ad_1^{SLALoM} = P_{\frac{K\phi}{\sqrt{G}}2^H}$. If A's home square is in the level-2 square B is in, then $Ad_2^{SLALoM} = P_{\frac{K\phi}{\sqrt{G}}2^H} \times P_{\phi 2^H}$.

CRLS

The FACE algorithm guarantees that that read and the write quorums will intersect if the network is connected. Therefore, the probability to retrieve A's location information is $Ad^{CR} = Pc$. If $r > r_0$, then $Ad^{CR} = 1$.

FSLS

If B and A are not in the same zone (i.e., faraway destination nodes), B sends a query packets to (β) zones. The average number of hops each query packet traverses to reach each of the β servers is $E(n_H)$. The probability that B accesses a single faraway server is: $P_{\phi 2^H}$, and the probability that at least one zone responds is:

$$Ad^{FSLS} = \sum_{k=1}^{k=\beta} \begin{pmatrix} \beta \\ k \end{pmatrix} P_{\phi 2^H}^k (1 - P_{\phi 2^H})^{\beta - k} = 1 - (1 - P_{\phi 2^H})^{\beta}$$

We can use a Taylor series approximation for Ad^{FSLS}, we obtain

$$Ad^{FSLS} = \beta P_{\phi 2^H}$$

B retrieves A's location information from the server of its current zone only if B and A are located in the same zone (i.e., nearby destination nodes), and the server is reachable. The probability that both nodes are in the same zone is $\frac{1}{G}$, and the average number of hops between two nodes in a zone is $(\frac{c_1 R}{z\sqrt{G}})2^H$, and therefore, $Ad^{FSLS} = P_{\frac{\phi}{\sqrt{G}} 2^H}$.

$$Ad^{FSLS} = \begin{cases} P_{\frac{\phi}{\sqrt{G}} 2^H} & \text{if B is in A's zone} \quad \text{(nearby destinations)} \\ \beta P_{\phi 2^H} & \text{if B is not in A's zone} \quad \text{(faraway destinations)} \end{cases}$$

6.2.9 Comparison

We will compare the availability degree of FSLS with that of HIGH-GRADE, GLS, DLM, SLURP, SLALoM, and CRLS when $\{P_i\}$ is a decreasing and an increasing sequence.

We note that $Ad^{FSLS} > Ad^{SLURP}$, and $Ad^{FSLS} < Ad^{CR}$ in both cases: decreasing and increasing sequence.

Case 1: $\{P_i\}$ is an increasing sequence

$P_{\phi 2^H} > P_{\phi 2^j}\ \forall\ 0 < j < H-1$, and hence Ad^{FSLS} is greater than Ad_j^{HG}, Ad_j^{GLS}, and $Ad_j^{DLM}\ \forall j \in [0, H]$.

In case of nearby destinations: $Ad_1^{SLALoM} > Ad^{FSLS}$. Thus, for $\sqrt{n}$ nodes, the probability that FSLS obtains A's location information is less than that of SLALoM.

Case 2: $\{P_i\}$ is a decreasing sequence

We assume that the FSLS's zone is less than the $level-(H-1)$ square, and greater than the level-0 square. For the sake of simplicity, FSLS's zone is depicted as level-1 square in figure 6.5.

If A is located in the same level-H as B (faraway destination), we have

$$Ad_H^{HG} = (P_\phi \prod_{j=1}^{j=H-1} P_{\phi 2^j}^2) \times P_{\phi 2^H}^2 = C_{HG} \times P_{\phi 2^H}^2$$

$$Ad_H^{GLS} = P_{\phi 2^H} \times \prod_{j=0}^{j=H-1} P_{\phi 2^H} = C_{GLS} \times P_{\phi 2^H}$$

$$Ad_H^{DLM} = (P_{\phi 2^k} \prod_{j=k}^{j=H-1} P_{\phi 2^j}) \times P_{\phi 2^H} = C_{DLM} \times P_{\phi 2^H}$$

Ad^{FSLS} is greater than Ad_H^{HG}, Ad_H^{GLS}, and Ad_H^{DLM}, since $\beta >$ is greater than C_{HG}, C_{GLS}, and C_{DLM}. Therefore, for $(\frac{3n}{4})$ nodes (i.e., 75% of nodes), the probability that FSLS obtains A's location information is higher than that of HIGH-GRADE, GLS, and DLM.

If A is located in B's zone (i.e., nearby destination), Ad^{FSLS} is greater than $Ad_{\frac{\phi}{\sqrt{G}}}^{HG}$, $Ad_{\frac{\phi}{\sqrt{G}}}^{GLS}$, and $Ad_{\frac{\phi}{\sqrt{G}}}^{DLM}$. A zone contains $\sqrt{n}$ nodes, So for $\frac{3\sqrt{n}}{4}$, the probability that FSLS obtains A's location information is higher than that of HIGH-GRADE, GLS, and DLM.

As for SLURP, we can easily deduce that the availability degree of SLURP is lower than that of FSLS.

As for SLALoM, we have two cases:

- In case of faraway destination nodes: $Ad_2^{SLALoM} < Ad^{FSLS}$.
- In case of nearby destination nodes, $Ad_1^{SLALoM} < Ad^{FSLS}$.

6.2.10 Discussion

Table 6.2: Location service performance comparison

	HIGH-GRADE	GLS	DLM	SLURP	SLALoM	CRLS	FSLS
Location update cost	$O(v \log n)$	$O(v\sqrt{n})$	$O(v\sqrt[3]{n})$	$O(v\sqrt{n})$	$O(v\sqrt[3]{n})$	$O(v\sqrt{n})$	$O(v\sqrt[4]{n})$
Location query cost	$O(\sqrt{n})$ (uniform) $O(\log n)$ (localized)	$O(\sqrt{n})$ (uniform) $O(\log n)$ (localized)	$O(\sqrt[3]{n})$ (both)	$O(\sqrt{n})$ (both)	$O(\sqrt[3]{n})$ (both)	$O(v\sqrt{n})$ (both)	$O(\sqrt{n})$ (uniform) $O(\sqrt[4]{n})$ (localized)
Storage cost	$O(n \log n)$	$O(n \log n)$	$O(n\sqrt[3]{n^2})$	$O(n\sqrt{n})$	$O(n\sqrt[3]{n})$	$O(n\sqrt{n})$	$O(n)$

Table 6.3: Number of nodes for which the location Availability of the service is less than that of FSLS

	HIGH-GRADE	GLS	DLM	SLURP	SLALoM	CRLS
Case 1	n	n	n	n	$n - \sqrt{n}$	0
Case 2	$\frac{3}{4}(n + \sqrt{n})$	$\frac{3}{4}(n + \sqrt{n})$	$\frac{3}{4}(n + \sqrt{n})$	n	n	0

We summarize the scalability metrics of the seven location services in Table 6.2. The different costs of HIGH-GRADE, GLS, DLM, SLURP, and SLALoM are provided in [70]. The update location cost in FSLS is asymptotically better than GLS, DLM, SLURP and SLALoM, and CRLS. It is also asymptotically better than HIGH-GRADE when $n \in [5, 5500]$. We observe that FSLS improves the query location cost over SLURP, GLS, DLM, and SLALoM. It reduces query latency by limiting the distance a query packet traverses. Location information of nodes residing in the same zone as the querying nodes can be obtained within time period lower than that of SLURP. Using the *unreachable zone list*, FSLS reduces the number of messages generated to perform an update or a query operation. Moreover, it can incurs low query cost (i.e., $\sqrt[4]{n}$) if the faraway servers are unreachable, and the querying and the queried nodes are in the same zone. FSLS outperforms the other five location services in terms of storage cost, since it is classified as a some-for-some approach.

Table 6.3 gives the number of queried nodes for which the location availability of each location service is less than that of FSLS. The two-level and hierarchical approaches gives lower availability degree than FSLS for the majority of queried nodes, because they impose on querying node to contact a chain of servers in a certain order. If one of these servers are unreachable, the query operation cannot being carried out, whereas in FSLS, it is sufficient that at least one server responds to get the location information of the queried node. SLURP gives low location availability in comparison with FSLS since it queries only one region. CRLS outperforms the

other location services in terms of location availability. However, this comes at the cost of increasing update, query, and storage costs.

6.2.11 Simulation results

We study the performance of the proposed service using GloMoSim simulator [153]. All nodes have a transmission range of 250 m. They move according to the waypoint mobility model. In this model, a node randomly selects a location and moves toward it with a constant speed uniformly distributed between zero and a maximum speed V_{max} , then it stays stationary during a pause time of 1 second before moving to a new random location. Initially, each mobile node has a battery capacity of 400 Joules. We have implemented three location services, which are: (1) CRLS [130], (2) HIGH-GRADE [151], and (3) FSLS. We have also implemented a geographic routing protocol GPSR [81].

Since the performance of FSLS is dependant on the selection of α, we decided to simulate three versions of FSLS, with α= 2, α=4 and α = 6 (hereafter called FSLS-2 and FSLS-4, and FSLS-6) respectively.

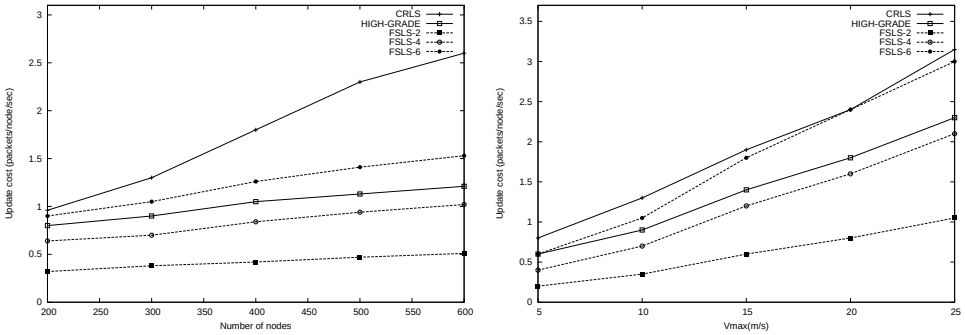
More than the update location cost and the query location cost defined in Section 6.2.6, the following metrics are evaluated for the location service protocols.

1. *Location availability:* is defined as the percentage of success queries over the total number of queries issued.

2. *Service lifetime:* The time until all the location servers run out of power.

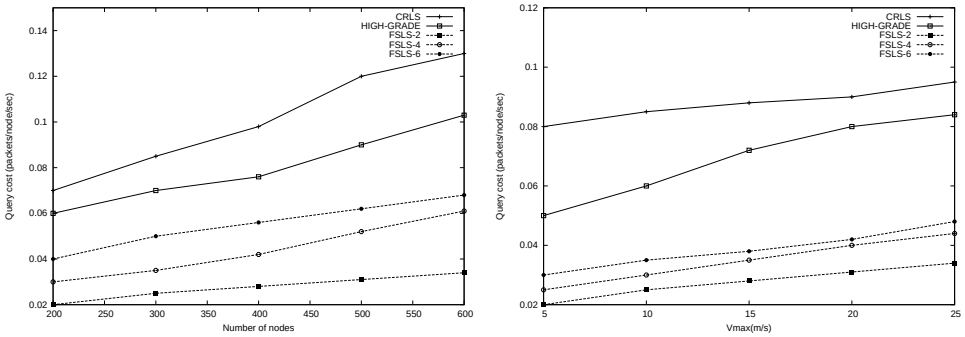
All of our simulations are conducted without any data traffic, which discard the factors affecting node lifetime, and hence allows us to better judge the performance of services. To study the performance of the service for network scalability and node mobility. We have carried out two sets of simulations. The first study fixes the node density to be 100 nodes/km2 and Vmax to be 10 m/s while varying he total number of nodes in the network. In FSLS, a network size of 2km^2 is divided into zones of 500m. The length of the zone is increased by 50m as the network length is increased by 1000m. The second study evaluates the location services in a square of 3km^2 consisting of 300 nodes by varying the average node speed.

Figure 6.6(a) shows the location update cost as a function of n. Location update cost is the number of location update packets that are originated at or are forwarded



(a) (b)

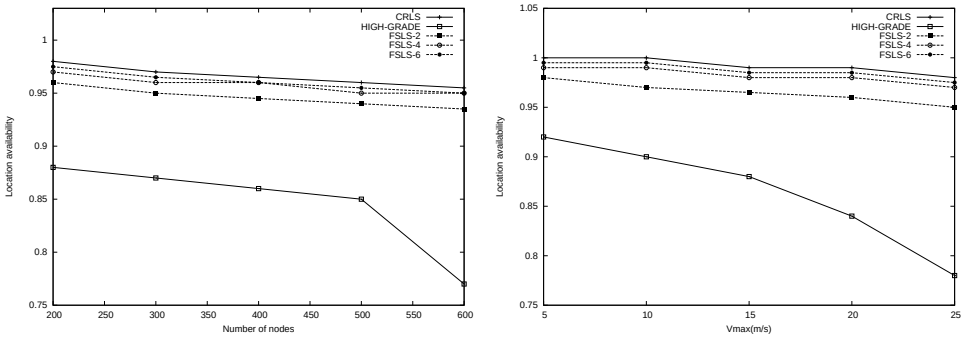
Figure 6.6: Update cost



(a) (b)

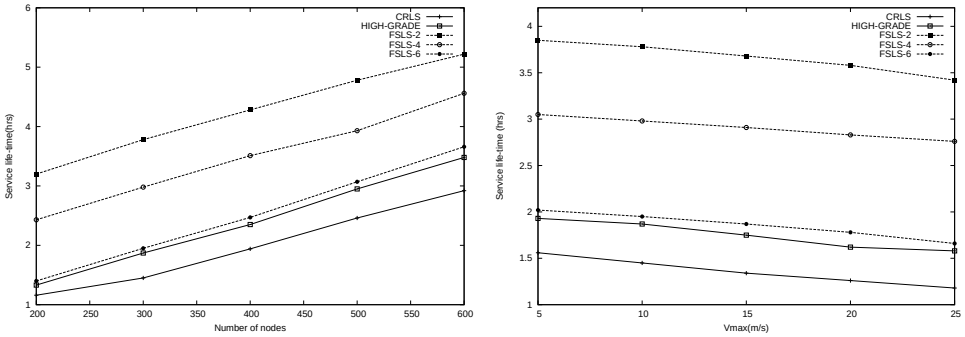
Figure 6.7: Query cost

by a node per second. CRLS has the highest overhead among all three protocols. This is primarily due to the fact that the length of rows and columns is increased. HIGH-GRADE has a lower number of updates than CRLS. However, it incurs more update than FSLS-2, and FSLS-4 because as network size is increased from 2km^2 to 6km^2, the hierarchical level H is increased from 3 to 5. In HIGH-GRADE, whenever a node crosses level-i square boundary, it needs to update all its level-j servers for $i \geq j - 1$, and as H increases, HIGH-GRADE needs to update more servers. In FSLS, as the network size is increased, the zone size is increased, which decreases the frequency at which each node generates an update packet, but update packets take longer paths which makes FSLS grows as n increases. Since the update cost of HIGH-GRADE increases only logarithmically with the number of nodes, it performs



(a) (b)

Figure 6.8: Location availability



(a) (b)

Figure 6.9: Service life-time

better than FSLS-6.

Figure 6.6(b) shows the location update cost as a function of maximum node speed. CRLS scales worse asymptotically than FSLS and HIGH-GRADE. This can be explained as follows: as nodes move more rapidly, the network topology changes more frequently, and CRLS generates update packets at a higher frequency than that of HIGH-GRADE and FSLS. HIGH-GRADE generates an update packet each time it crosses a level-0 square boundary (i.e., side length is 182.5 m), whereas FSLS generates an update packet each time it crosses a zone boundary (side length is 600m). We can notice that the update frequency of FSLS is less 3 times than that of HIGH-GRADE. For this reason, FSLS-2 and FSLS-4 perform better than HIGH-GRADE, but FSLS-6 performs worse than HIGH-GRADE since it sends each update packet to six zones.

Figure 6.7(a) shows the location query cost as a function of n. Location query cost is the number of location query packets that are originated at or are forwarded by a node per second. CRLS has the highest overhead among all three protocols for the same reason discussed in reference to figure 6.6(a). HIGH-GRADE transmits more query packets than FSLS because the majority of the queried nodes are not close to the querying node, and the query packet has to travel high up and low down in the hierarchy to retrieve the queried node's location information. On the other hand, FSLS tries to reduce the number zones targeted by each query operation.

Figure 6.7(b) shows the location query cost as a function of maximum node speed. From the figure, we can notice that node speed has much effect on the query cost. This validates our analytic results.

Figure 6.8(a) shows the location availability as a function of n. CRLS outperforms FSLS and DLM because the intersection of row and column is guaranteed if the network is not partitioned. FSLS outperforms HIGH-GRADE because its location servers replicates their location tables onto a new node before they leave their zones or run out of power. This technique improve the service reliability. The location information in this case is more likely to be available in the zone. In contrast to FSLS, HIGH-GRADE does not deal with the depletion of node's power. Moreover, To obtain the exact location of a node, HIGH-GRADE has to access a chain of location servers in a certain order. If any single square in this chain is empty, the query operation fails. The location availability of CRLS is slightly affected by increasing number of nodes, whereas that of FSLS and HIGH-GRADE decreases as n increases. This is due to the fact that the query packets in FSLS takes longer routes and that in HIGH-GRADE has to reach more location servers. We can also notice, that as α increases, the location availability of FSLS also increases. Figure 6.8(b) shows the location availability as a function of maximum node speed. Location availability of the location services decreases as n increases because it might happen that query packets do not reach their location servers, or some servers have stale information because of missing location information on these servers.HIGH-GRADE is more affected by node mobility than the other protocols, because the mobility of location servers of a certain level increases query failures. To deal with this case, the departed server replicates its location information on another node.

Figure 6.9 shows the service lifetime experienced by the location services. As

CRLS generates more packets, its location servers will die earlier than those of HIGH-GRADE and FSLS. FSLS performs better than HIGH-GRADE because the location servers replicate their location tables onto a new elected location server before they disappear from their zones, which considerably increases the lifetime of FSLS.

6.3 Location-based data replication schemes

In the following schemes, we consider a system consisting of a set of n nodes. Each node i has a data item D_i associated with it. We assume that the time between updates to any data item D_i follows an exponential distribution with mean $1/\mu_i$, and each node i generates a query according to a Poisson distribution with mean rate of λ_i.

6.3.1 Flat-based Data Replication Scheme (FDRS)

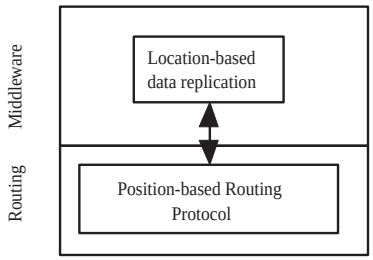


Figure 6.10: FDRS architecture

We propose the Flat-based data Replication Scheme (FDRS) (see Figure 6.10), is a replication protocols that runs on top of a position-based routing protocol. FDRS is an enhanced version of Rendezvous Regions protocol (RRs) [135]. In RRs, each data item is mapped to a unique region, and inside the region many nodes store the same information. The drawback of this design is it does not handle the case of empty regions, which significantly degrades data availability. To reduce the risk of sending update and query packets to empty regions, we propose to use a known hash mapping algorithm that maps each data item x to a set H_x, which consists of fixed α regions. Figure 6.11 depicts the partitioning scheme, in which the respective regions' identifier are shown in the upper-left corner of each region, the replica servers are colored black, and $H_A = \{5, 14\}$. Only a unique node from each region is selected

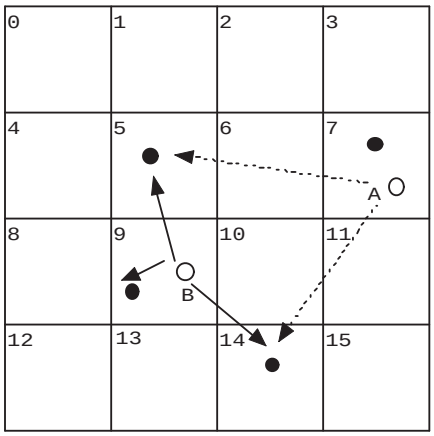


Figure 6.11: Area partitioned according to the flat-based approach

as a replica server. The data replication protocol performs the update and the query operations, it also handles the cases of servers leaving their regions, or empty regions in same way as the protocol presented in Section 6.2 does.

6.3.2 Hierarchical-based Data Replication Scheme (HDRS)

We propose the Hierarchical-based data Replication Scheme (HDRS)(see figure 6.12). HDRS needs to implement a location service [151] so as to locate its server replicas. The role of a location service is to map the ID of a node to its geographical position. Each location service performs two basic operations: the *location update* and the *location query*. The location update is responsible for replicating information about the current location of a given node A to a set of nodes called *location servers*. If a node B wants to know the location of A, it sends a location query message to some or all the location servers of A. As a location service, we have chosen to use HIGH-GRADE [151]. In [152], it has been demonstrated that HIGH-GRADE is the most scalable location service. An overview of HIGH-GRADE can be found in Chapter 3.

In the hierarchical-based data replication scheme, two types of information are used. The first ones are the data items that are handled by the replication protocol, and the second ones are the location information of the nodes that are handled by HIGH-GRADE. HDRS uses a position-based routing protocol to forward its update and query packets. The replica server that stores the data item of each node A is

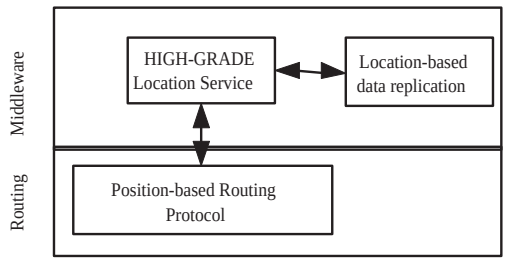


Figure 6.12: HDRS architecture

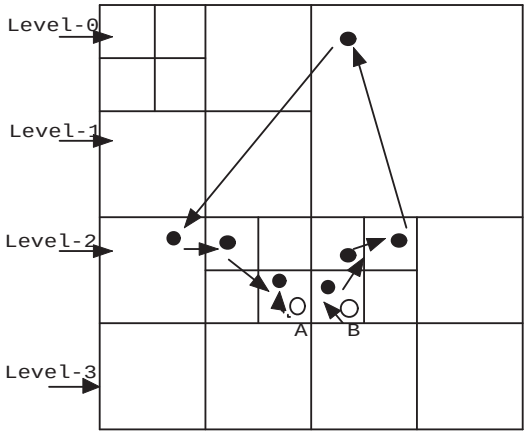


Figure 6.13: Area partitioned according to the hierarchical-based approach

$LSP_{A,0}$. In figure 6.13, the black circles denote the replica servers. When A crosses the boundary of its level-0 square, it sends its new position along with its data item to its replica server. In addition, when D_A is updated, the new version of D_A is also sent to $LSP_{A,0}$ (figure 6.13). When a node B wants to find A's data item, it contacts the HIGH-GRADE location service that locate the replica server of A as explained in Chapter 3.

6.3.3 Analytical study

We compare FDRS, HDRS, RRs, and a topology-based data replication protocol like the quorum-based protocols [82, 87] in terms of data availability and scalability. The scalability metric consists of four costs: *Update cost (c_u)*, Query cost (c_q), Replication cost (c_r), and *Storage cost (c_s)*. To make fair comparison, we consider that the quorum size is α. We assume that each node generates update and query packets at rate λ

Table 6.4: Comparison of data replication protocols

	HDRS	FDRS	RRs	topology-based
Replication cost	$O(n)$	$O(\sqrt{n})$	$O(n)$	$O(\sqrt{n})$
Update cost	$O(v \log n + \mu)$	$O(\alpha\mu\sqrt{n})$	$O(\mu\sqrt{n})$	$O(\alpha\mu(n+\sqrt{n}))$
Query cost	$O(\lambda\sqrt{n})$ (uniform) $O(\lambda \log n)$ (localized)	$O(\alpha\lambda\sqrt{n})$ (uniform) $O(\lambda \sqrt[4]{n})$ (localized)	$O(\lambda\sqrt{n})$ (both)	$O(\alpha\lambda(n+\sqrt{n}))$ (both)
Storage cost	$O(\log n)$	$O(\alpha\sqrt{n})$	$O(\sqrt{n})$	$O(\alpha\sqrt{n})$
Availability	P^{2j+1}	P_{FDRS}	P	P_{FDRS}

and μ respectively.

Update cost

In FDRS, the area of each region is a. Thus, the ad hoc network is divided into $G = (\frac{S}{a})$ regions. We assume that The cost of broadcasting in a square by a node, b, is proportional to the number of transmissions needed to cover the said square. Thus, $b = O(a)$. The average distance between two random points in a square of area A (i.e., u) is proportional to $\sqrt{S}$ [45], and S is proportional to n.

When a node updates its data item, it generates an update message that is sent to α regions. When the update packet reaches a node of one of those regions. If that node has not a cached route to the replica server, it broadcasts a route discovery packet to all nodes within the region to establish a route to the server. The cost of updating one server is $(b + u)$.

The update cost can be written as:

$$c_u = O(\mu\alpha(b + u)) \text{ packets/sec/node}$$

Substituting u and b, we have

$$c_u = O(\mu\alpha(\sqrt{n} + a)$$

If we consider that a is proportional to $\sqrt{n}$, thus the update cost of FDRS is $O(\mu\alpha\sqrt{n})$ packets/sec/node

In [152], it is shown that the location update cost of HIGH-GRADE is $O(v \log n)$. Moreover, when D_i is updated, the update packet is sent to the server in its level-0 square. The side length of a level-0 square, R is assumed to be constant. Thus, the update cost of HDRS is $c_u = O(v \log n + \mu)$ packets/sec/node.

Query cost

In FDRS, when a node wishes to find the data item of a specific target node D, it first sends a query packet to its current region. The query packet traverses a distance of $O(\sqrt{a})$. So, the query cost for localized traffic is $O(\lambda\sqrt[4]{n})$ packets/sec/node.

If the node receives no response, it sends a query packet toward the center of each region $\in H_D$. So, the query cost for uniform traffic is $O(\lambda\sqrt{n})$ packets/sec/node

From [11], we can derive that the query cost of HDRS is $O(\lambda_i\sqrt{n})$ packets/sec/node for uniform traffic, and $O(\lambda_i \log n)$ packets/sec/node for localized traffic.

Replication cost and storage cost

In FDRS, there are $(\frac{S}{a})$ servers in the network. So, $c_r = O(\sqrt{n})$. Each server stores $(\frac{\alpha \times n}{G})$ data items, and there exist G replica servers. Thus, the storage cost is $O(\alpha \times \sqrt{n})$.

In HDRS, one server must exist in each level-0 square. It means that there are 4^H replica servers. As H is proportional to $\log(\sqrt{n})$, $c_r = O(n)$. In [152], it is shown that each server in HIGH-GRADE stores $O(\log n)$ location information. Moreover, each replica server stores the data items of nodes that are located its level-0 square. As there are $(\frac{n}{4^H})$ nodes in each level-0 square, each replica server stores $O(1)$ data items.

Data availability

In [47], the probability P that two nodes are connected is estimated. In RRs, as a node sends its query to a single region, so its data availability is P. In FDRS, the query is sent to α regions, and the probability that at least one region responds is:

$$P_{FDRS} = \sum_{k=1}^{k=\alpha} \left(\begin{array}{c} \alpha \\ k \end{array} \right) P^k (1-P)^{\alpha-k}$$

In HDRS, if A, and B are located in the same level-j square. As described previously, each query packet is forwarded sequentially from $pLSP_0$, $pLSP_1$, ..., up to $pLSP_j$ (i.e., LSP_j), then to LSP_{j-1} until LSP_0 is reached. Thus, the data availability of HDRS is: P^{2j+1}.

In the quorum-based replication protocol, the query is sent to α nodes. We assume that there is $\beta < \alpha$ nodes in the intersection between the update quorum and the

query quorum. Therefore, the probability that at least one node responds is:

$$P_{FDRS} = \sum_{k=1}^{k=\beta} \left(\begin{array}{c} \beta \\ k \end{array} \right) P^k (1-P)^{\beta-k}$$

Comparison and Discussion

We summarize the performance metrics of FDRS, HDRS, RRs, and the topology-based replication protocol in Table 6.4. The topology-based replication protocol shows poor scalability, because the topology-based routing protocol needs to flood the route request packet in the whole network to locate the target node, which requires a cost of $O(n)$, and the route reply packet traverses a distance of $O(\sqrt{n})$. HDRS is asymptotically better than FDRS and RRs in terms of update and query cost. On the other, the update cost of FDRS and RRs is asymptotically better than that of HDRS when $\mu_i < v(\frac{\log n}{\sqrt{n}})$. In addition, the query cost of FDRS for localized traffic is asymptotically better than that of HDRS when $n \in [5, 5500]$.

In terms of replication cost, FDRS and the topology-based protocol are more scalable than HDRS and RRs, because HDRS needs that a server to be available in each level-0 square, while RRs needs to elect many servers in each region. This leads at worst case to a complexity of $O(n)$. On the other hand, HDRS outperforms the other protocols in terms of storage cost. The probability that B in FDRS accesses A's data item is higher than that of HDRS. FDRS and offers higher data availability than HDRS, because HDRS imposes on querying node to contact a chain of servers in a certain order. If one of these servers is unreachable, the query operation cannot being carried out. The data availability of RRs is low because it queries only one region, whereas in the case of FDRS, it is sufficient that at least one server among α responds to get the data item of the queried node. As $\beta < \alpha$, the data availability of the topology-based replication protocol is lower than that of FDRS. From this study, we can conclude that HDRS works efficiently in large-scale ad-hoc networks with high update rates and localized traffic, while FDRS shows the best performance in large-scale ad-hoc networks with low update rates and uniform traffic.

6.4 Conclusion

In this chapter, we have present a location service and two location-based data replication protocols, aiming at making tradeoffs between scalability and availability.

The location-based protocol (FSLS) is a new location service aiming at making tradeoffs between scalability and location availability. FSLS combines the hash-based sets system and a flat hashing-based structure to select zones where location information will be stored. A new server is elected when the current server is about to leave the zone or run out of power, which improves the service reliability, and increases significantly the service lifetime. We have analyzed the scalability and the availability of FSLS as well as six other location services. Analysis has shown that FSLS offers a good trade-off between location availability and scalability. It comes second after CRLS [130] in terms of location availability, and it is the closest competitor to HIGH-GRADE [151] in terms of scalability, since its communication cost can scale as $O(v\sqrt[4]{n})$, especially under localized traffic patterns. The analysis results have been further supported by simulation experiments.

As far as the location-based data replication protocols are concerned, our analysis shows that the design choice and the rate of data update affect the scalability and the availability of the protocols. HDRS shows low update cost, query cost, and storage cost. FDRS outperforms HDRS in terms of replication cost and data availability. FDRS can be more scalable than HDRS in terms of update when the update rate is low. Moreover, the query cost of FDRS for localized traffic is asymptotically better than that of HDRS for some values of n. From this study, we can conclude that HDRS works efficiently in large-scale ad-hoc networks with high update rates and localized traffic, while FDRS shows the best performance in large-scale ad-hoc networks with low update rates and uniform traffic.

Chapter 7

K-hop Cluster-based data replication protocol

7.1 Introduction

The objective of a clustering algorithm is to find a feasible interconnected set of groups covering the entire node population. Clustering helps to reduce the overhead due to generation and propagation of information, and efficient network management. A set of nodes S in $G = (V, E)$ is called a L-hop dominating set if every node in V is at most $L(L > 1)$ hops away from a vertex in S. For a graph G and an integer $K \geq 0$ the problem of determining whether G has dominating set of size $\leq K$ was proven to be NP-complete [4, 48]. Polynomial time and message complexity approximation solution to the K-clustering problem (where every two nodes in the cluster are at most K hops away from each other) can be found in [48]. In this chapter, we will present a new clustering algorithm [35] that can limit the query delay and predict group partitioning based on the residual link and node lifetime.

7.2 Stable K-hop DAG Algorithm

We propose the *Stable K-hop DAG Algorithm*, which aims at finding nodes that are more likely to be connected for a long time period. To achieve this, we propose to use the residual node-link lifetime as presented in Chapter 5. The residual node-link lifetime between two connected nodes i and j is given by the following equation.

$$\chi_{ij} = \min\left(Tpow_i, Tpow_j, Tmob_{ij}\right) \tag{7.1}$$

where:

$$Tpow_i = \frac{E_i}{\max\left(R_i\right)}(s)$$

$$Tmob_{ij}(t) = \frac{S_{ij}^m(t) - S_R}{\frac{1}{N-1}\sum_{k=m-N+1}^{m}|V_{ij}^k(t)|}$$

The definition of $S_{ij}^m(t)$, S_R, and $V_{ij}^k(t)$ can be found in Chapter 4.

Each node i checks if the predicate $P \equiv (\chi_{ij} - \Delta t < \chi_{th})$ holds true. If so, it considers that the link (i, j) is about to break, and hence the link is called *weak*. Otherwise, it is called *strong*. χ_{th} denotes the lower-bound of the residual link lifetime at which node i has to generate a warning and perform preemptive actions like data replication.

As discussed in Chapter 3, the clustering algorithms offer poor scalability. To increase their scalability, we propose to construct clusters consisting of nodes that are more likely to be connected for a long time period. This characteristic has two advantages: (1) we reduce considerably the cost of cluster maintenance, and (2) we can predict cluster partitioning before its occurrence. To achieve this, we propose the *Stable K-hop DAG algorithm*. Before presenting the detailed description of the proposed clustering algorithm, we first define the following terms:

Definition 20. *The node for which the predicate $Q \equiv (Tpow_i - \Delta t < \chi_{th})$ holds true is called short-lived node. Otherwise, it is called long-lived node.*

Definition 21. *Node j is called a strong neighbor of node i if $Q \equiv (\chi_{ij} - \Delta t < \chi_{th})$ holds true. Otherwise, it is called a weak neighbor.*

Definition 22. *A path which is composed only of strong links is called* ***strong path****. Otherwise, it is called* ***weak path****.*

Definition 23. *A Directed Acyclic Graph (DAG) is a a graph with no cycles and each node has a directed path toward a sink node.*

Definition 24. *A K-hop directed acyclic graph (K-DAG) is a DAG, in which each node is at most K hops away from the sink node.*

Definition 25. *A Stable K-hop directed acyclic graph (Stable K-DAG)*[1] *is a K-hop DAG, in which each node has a strong path toward the sink node.*

Each node i maintains three variables, the Group Index (GI_i), the height (H_i), and the Hop variable. The group index GI_i is a couple (Tc_i, GL_i), where GL_i is the identifier of the node considered to be the group leader of node i. Tc_i denotes the time at which the group leader has started the creation of its stable K-hop DAG. GI_x is said to have lower priority than GI_y, and we write $GI_x \prec GI_y$ iff

$$((Tc_x > Tc_y) \vee (Tc_x = Tc_y \wedge GL_x > GL_y))$$

Like in Chapter 5, the hight H_i =(lid_i, τ_i, oid_i, r_i, δ_i, i), and a new reference level is generated when a node i loses all its strong outgoing links toward its group leader.

The *stable K-hop DAG algorithm* consists of two logical parts: The first deals with the formation of groups, and the second deals with dynamically reconfiguring the groups to take into account group partitioning and groups merging. We assume that the time between updates to any data item D_i follows an exponential distribution with mean $1/\mu_i$, and each node i generates a query according to a Poisson distribution with mean rate of $\lambda_i = \sum_{i=1}^{n} \lambda_{ij}$, where λ_{ij} denotes the mean rate at which node i generates a query to D_j. Nodes have synchronized clocks through an appropriate algorithm [129] that guarantees an upper bound on the clock error between any two pair of nodes in the network.

7.2.1 Localized stable K-hop DAG creation

Initially, each node i sets GI_i, and H_i to null. As shown in Algorithm 9, the construction of the stable K-hop DAG can be started by any node i whose H_i is null (Algorithm 9(A)). It sets GI_i, H_i, and Hop to (t, i), $(0, 0, 0, 0, i)$, and 0 respectively, where t represents the current time. Then, it broadcasts a *CreateDag* message containing its group index, its height, and the Hop variable. Upon receiving such a message, each node j calls the *JoinDag* procedure. In this procedure, if $H_j = null$, j distinguishes between two cases: (i) $Hop < K$, then j joins the DAG, and sends the tuple $< D_j, \mu_j >$ to GI_j, and (ii) $Hop \geq K$, then it declares itself a group leader and

[1]Throughout this chapter, the terms "group", "cluster" and "Stable K-DAG" are used interchangeably.

constructs its stable K-hop DAG. Otherwise if i and j have different group indexes, the one that has a lower group index (i.e., j) and its distance to the new group leader GL_i is less than K, joins the new stable K-DAG, and sends its data item D_j along with μ_j to GI_j. It is obvious that more than one node can concurrently start constructing its stable K-hop DAG. When multiple stable K-DAGs meet, the optimal choice is to not modify an older Stable K-DAG. So, we stop the propagation of the current *CreateDag* messages. The node that changes its group index, broadcasts an *UpdateDag* message containing its older leader group, its current group index, its height and the variable *Hop*. Upon receiving such a message, a node k that finds that its group index is deleted, sets its GI_k and H_k to null and calls the *JoinDag* procedure.

After a predefined timeout, GL_i calculates χ_{th} and propagates this information in its group. Each node i compares χ_{ij} to χ_{th} to know if the links (i, j) are strong or weak, and if i is long-lived or short-lived. χ_{th} is the lower-bound of the appropriate time to trigger the reference level propagation in case a long-lived node loses all its strong links (see Algorithm 10(A)). Node that starts a new reference level needs that the residual lifetime of the last strong outgoing link be sufficient to predict the K-hop DAG partitioning and to download a copy of the database stored at the group leader. The group leader calculates χ_{th} so that the following predicate (χ_{th} >*time to replicate a database+time to predict the partitioning*) must hold. The database replication process is performed in at most $(K \times \sum_{j \in G_i} |D_j|/Bw)$ seconds, where $\sum_{j \in G_i} |D_j|$ is the size of data items owned by the members of the group G_i, and which need to be replicated and transmitted over an end-to-end wireless connection with a bandwidth of Bw bits/s. In [109], it is shown that at worst case, we need $O(2l)$ units of time to detect a network partitioning, such as l is the length of the longest directed path in the network segment affected by a topological change. The K-hop DAG maintenance algorithm needs the same complexity of time to predict the partitioning of the K-hop DAG. If $\overline{X}$ is the average time to send the *Failure* message over a wireless link, the lower bound of χ_{th} is: $K(2\overline{X} + \sum_{j \in G_i} |D_j|/Bw)$.

Figure 7.1 shows the result of executing Algorithm 9 on a network consisting of 33 nodes $(M_1, \cdots, M_{33})$. In this figure, six stable 2-hop DAGs are created. M_1, M_{11}, M_{18}, M_{23}, M_{26}, and M_{32} are group leaders. The black circles and white circles denote the group leaders and the regular nodes respectively. Each regular node has a directed

path toward its group leader. The weak links and the short-lived nodes are shown as dotted lines and dotted circles respectively. Each group leader maintains the list of adjacent groups via some nodes called *frontier nodes* denoted by gray circles in figure 7.1, and which are connected to other groups. The frontier nodes inform their group leaders about the appearance and the disconnection of adjacent groups.

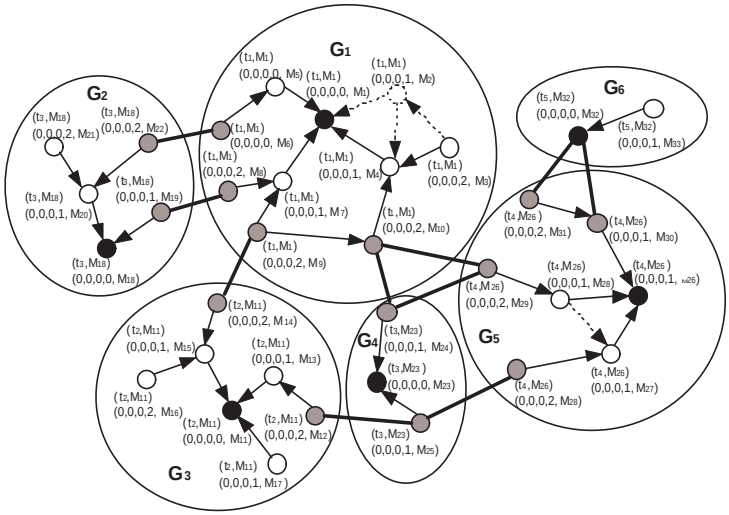


Figure 7.1: An example of executing the stable K-hop DAG Construction algorithm

7.2.2 Localized stable K-hop DAG maintenance

The stable K-hop maintenance algorithm is triggered either when a long-lived node i loses its last strong outgoing link toward the group leader (Algorithm 10(A)), or when two groups merge (Algorithm 10(F)).

7.2.3 Cluster prediction algorithm

Algorithm 10 is executed as follows: If $SN_i = \phi$ (i.e., SN_i is the set of nodes which i has a strong connection with them), it declares itself a group leader and starts reconstructing its own stable K-hop DAG. Otherwise, it defines a new reference level $(t, i, 0)$, sets the disconnected set ($disc\_set_i$) to $\{i\}$, and broadcasts a *Failure* message containing its height and its disconnected set. The *Failure* message is propagated in the graph G_i obtained by removing the short-lived nodes and the weak links in G_i.

Algorithm 9 Stable K-hop DAG Construction

Procedure $JoinDag(GI_j, H_j, Hop)$
1: **if** $(H_i = null \wedge Hop < K)$ **then**
2: $GI_i = GI_j$; $H_i := (0, 0, 0, \delta_j + 1, i)$;
3: *Send* $< D_i, \mu_i >$ *to* GL_i;
4: *broadcast CreateDag*$(GI_i, H_i, Hop + 1)$;
5: **else if** $H_i = null \wedge Hop \geq K)$ **then**
6: $GI_i := (t, i)$; $H_i := (0, 0, 0, 0, i)$;
7: *broadcast CreateDag*$(GI_i, H_i, 0)$;
8: **else if** $(GI_i \prec GI_j \wedge Hop < K)$ **then**
9: $old = G_i$; $GI_i := GI_j$; $H_i := (0, 0, 0, \delta_j + 1, i)$;
10: *Send* $< D_i, \mu_i >$ *to* GL_i;
11: *broadcast UpdateDag*$(old, GI_i, H_i, Hop + 1)$;
12: **end if**

Case A: *(a node i wants to be a group leader* $\wedge H_i = null$*)*
1: $GI_i := (t, i)$; $H_i := (0, 0, 0, 0, i)$; $Hop = 0$
2: *broadcast CreateDag*(GI_i, H_i, Hop);

Case B: *(i receives CreateDag*(GI_i, H_i, Hop)*)*
1: $JoinDag(GI_j, H_j, Hop)$;

Case C: *(i receives UpdateDag*(old, GI_j, H_j, Hop)*)*
1: **if** $(GI_i = old)$ **then**
2: $GI_i := null$; $H_i := null$;
3: **end if**
4: $JoinDag(GI_j, H_j, Hop)$;

The aim of the reference level generation is to find if there exists an alternative strong path toward the group leader. In Algorithm 10(B) and Algorithm 10(C), a node i that loses all its strong outgoing links due to reference level propagation, decides locally to reflect its links. In this manner, the reference level is propagated until it is either stopped by a node that has a strong outgoing link or node i that has first defined the reference level, finds that all its strong neighbors $j \in SN_i$ have the same reference level and their $r_j = 1$ (Algorithm 10(D)). In this case, it decides locally that its subgraph G_i, which equals to *disc_set*, has no longer a strong path toward its group leader. So, it (1) declares itself a group leader, (2) starts constructing its new stable K-hop DAG, and (3) downloads the database held by its previous group leader.

Algorithm 10(F) shows the actions performed by a node i when it detects a neighbor j with different group leader. If GI_i has lower priority than GI_j, and node i is less that k hops away from GL_j, it becomes a member of GL_j's group, and broadcasts an *UpdateDag* message.

Algorithm 10 Stable K-hop DAG maintenance

Case A: When *a long-lived node i loses its last strong outgoing link*
1: **if** $SN_i = \phi$ **then**
2: $old := GL_i$; $GI_i := (t, i)$; H_i :=(0, 0, 0, 0, i);
3: ***download*** *the data items from old*;
4: **else**
5: H_i :=(t, i, 0, 0, i); $disc\_set := \{i\}$;
6: ***send*** $Failure(H_i, disc\_set)$; {*i starts a new reference level*}
7: **end if**

Case B: When *i has no strong outgoing links due to a link reversal following reception of a* $Failure(H_i, disc\_set)$ *message and the ordered sets* (τ_j, oid_j, r_j) *are not equal for all* $j \in SN_i$
1: $(\tau_i, oid_i, r_i) := \max\{(\tau_j, oid_j, r_j) | j \in N_i\}$
2: $(\delta_i, i) := (min\{\delta_j | j \in N_i \; with (\delta_j, oid_j, r_j) = (\delta_i, oid_i, r_i)\} - 1, i)$;
3: $disc\_set := disc\_set \cup \{i\}$;
4: ***send*** $Failure(H_i, disc\_set)$;

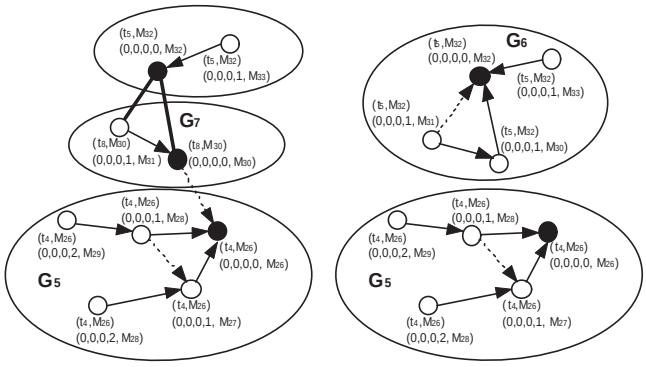
Case C: When *i has no strong outgoing links due to a link reversal following reception of a* $Failure(H_i, disc\_set)$ *message and the ordered sets* (τ_j, oid_j, r_j) *are equal with* $r_j = 0$ *for all* $j \in SN_i$
1: $H_i := (s_i, \tau_j, oid_j, 1, 0, i)$; $disc\_set := disc\_set \cup \{i\}$;
2: ***send*** $Failure(H_i, disc\_set)$;

Case D: When *i has no strong outgoing links due to a link reversal following reception of a* $Failure(H_i, disc\_set)$ *message and the ordered sets* (τ_j, oid_j, r_j) *are equal with* $r_j = 1$ *for all* $j \in SN_i$ *and* $i = oid_j$
1: $old := GL_i$; $GI_i := (t, i)$; H_i :=(0, 0, 0, 0, i);
2: ***download*** *data items from old*;

Case E: When *i has no strong outgoing links due to a link reversal following reception of an Update message and the ordered sets* (τ_j, oid_j, r_j) *are equal with* $r_j = 1$ *for all* $j \in SN_i$ *and* $oid_j \neq i$
1: H_i =(lid_i, t, i, 0, 0, i)

Case F: When *(a new strong link connects i and j)*
1: **if** $(GI_i \prec GI_j \wedge \delta_i < K)$ **then**
2: $old := G_i$; $GI_i := GI_j$; $H_i := (\tau_j, oid_j, r_j, \delta_j + 1, i)$;
3: *Send* $< D_i, \mu_i >$ *to* GL_i;
4: *broadcast UpdateDag*(old, GI_i, H_i);
5: **end if**

Figure 7.2 shows an example of executing Algorithm 10 on the network of figure 7.1. In figure 7.2(b), node M_{30} loses its last strong outgoing link. It generates a new reference level, node M_{31} executes *Algorithm 10(C)*. Upon receiving the *Failure* message sent by M_{31}, node M_{30} executes *Algorithm 10(D)* and detects that there is no strong path toward its group leader M_{26} (i.e., it predicts group partitioning). So, it declares itself group leader and constructs the group G_7. In figure 7.2(b), The *CreateDag* message generated by M_{30} will be stopped by M_{32} since ($GI_{M_{31}} \prec GI_{M_{32}}$). Nodes M_{31} and M_{30} executes *Algorithm 10(F)* and *Algorithm 9(C)* respectively, and hence become members of the group G_6.



(a) A time t_8 (b) A time $t_8 + h$

Figure 7.2: An example of executing the stable K-hop DAG maintenance algorithm

7.3 Localized hybrid data delivery protocol

In data delivery mechanisms, data can be obtained in two ways: (1) a push-based approach where a server periodically broadcasts the data items that it holds, and each client node can access a data item by waiting for the next broadcast period of the data item, and (2) pull-based approach where client nodes query the server for the data item they need, and hence the server separately responds to each query sent by each client, which increases network and server loads. Moreover, the query in the pull-based approach may need to traverse a long path to retrieve a data item. As the network size increases, query latency also increases, which represents an unscalable solution. The push-based approach can be seen as a scalable solution: a single broadcast can satisfy all pending requests for a given data item. Moreover, it is characterized by its higher throughput for data access in environment with a large number of clients, since the absence of channel access contention among clients increases considerably the available bandwidth and reduces channel access delay. However, the push-based approach suffers the disadvantage of additional control traffic that is needed to continually update invalidated data items even though no clients are querying them. This wasted effort can cause scarce bandwidth resources to be wasted. In the following, we propose a data delivery method that combines the push-based the pull-based approaches.

7.3.1 Localized pull-based data delivery

Each node is at most K hops away from a group leader. We propose to implement a pull-based approach, where each node queries its group leader for the data item it needs. This construction permits: (1) to apply a localized data retrieval since nodes can query only its nearby group leader for any data item, and (2) to considerably reduce query latency.

7.3.2 Localized push-based data replication protocol

Data replication protocols differ in the way they update their replicas. In the Single-item Broadcast Replication (SBR) policy, when D_i is updated, i broadcasts the new version of D_i to other nodes in the network. In [64], using only local information, each node independently decides whether it caches a data item. However, SBR and the protocol proposed in [64] suffer from the following drawback: if data items are updated at high rate, the throughput for data update may reduce due to channel access contention. In the Full Broadcast Replication (FBR) policy, when D_i is updated, i broadcasts its local copy of the whole database DB_i. When a node j receives this broadcast, j updates its version of D_i, and its local copy of each other item D_k, for which the version number in DB_i is more recent. Although this policy increases the consistency of local databases, but this comes at the price of an increase in update cost since the broadcast message is n times longer than that of SBR policy.

In [146], the Adaptive Broadcast Replication (ABR) is proposed. ABR adopts a lazily updating policy, in which when i updates D_i, it first determines whether the change in D_i justifies its broadcast. If so, node i broadcasts a message that contain D_i with a set S of data items from its local database. In order to do so, i estimates for each j and k the expected benefit to node k of including in the broadcast message its local version of D_j. For this estimation i maintains a knowledge-matrix. However, this solution is globalized since each node needs to maintain data items about all other nodes.

Motivated with these observation, we propose a localized data replication protocol that balance between data consistency and update cost. Each time node i updates its data item D_i, it immediately sends the new value of D_i to its group leader GL_i. Upon receiving such an update, the group leader does not perform an immediate data replication, instead it applies a lazy (delayed) data replication approach. Every

ΔT_{GL_i}, the group leader sends to its adjacent group leaders an update message containing all data items that was changed during the past ΔT_{GL_i}. The update packet is first sent to the frontier node connected to the adjacent group, then it is forwarded to the adjacent group leader. We remark that the update message is not flooded in the network as other cluster-based replication protocols do. Instead, it is sent directly to the adjacent group leaders along directed links. These group leaders in turn send the update message toward their adjacent groups and so on. This approach reduces the replication cost since it assembles a number of data items in one update message, but it increases the outdated data items returned by query operations. To make a good balance between cost and consistency, and knowing the time between successive updates of its group member, the group leader locally calculates the *update propagation period* ΔT_{GL_i} as follows:

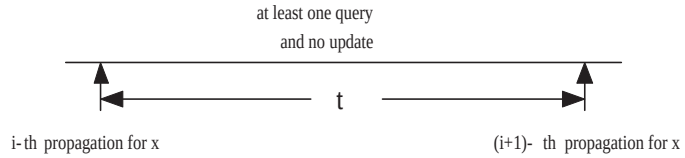
$$\Delta T_{GL_i} = \frac{\sum_{j \in G_{GL_i}} (1/\mu_j)}{|G_{GL_i}|} \tag{7.2}$$

where $|G_{GL_i}|$ is the number of nodes in the group. When a group leader i receives an update message, it updates the outdated data items and sends in turn the message to its adjacent group leaders. As nodes have limited storage capacity, group leaders only maintain the most queried data items. To do so, it calculates for each data item D_k the access frequency $f_k = \sum_{j \in G_{GL_i}} \lambda_{jk}$.

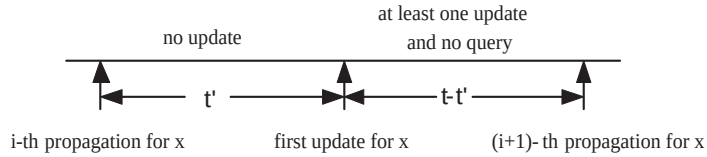
7.4 Performance analysis

In this section, we use a common theoretical framework to compare the scalability, data availability, and data accuracy of: (1) our localized replication scheme, (2) a partition-aware replication protocol (Derhab's protocol [36]), (3) a location-based replication protocol (RRs [125]), and a cluster-based replication protocol (CDRA [154]). To judge the merit of these protocols, we use the following metrics: *Replication cost (c_r)*, *Update cost (c_u)*, *Query cost (c_q)*, *Storage cost (c_s)*. We also define the following metrics, which are:

- *Availability degree (Ad):* The probability to access data items when needed.
- *Accuracy degree (Ac):* The probability to access data items when needed, and to acquire the most recent version of the data.



(a)



(b)

Figure 7.3: Cases for which a query operation succeeds

7.4.1 Analytical model

Under this framework, n nodes are randomly distributed in a region of area A. The average degree of a node γ (i.e., node density) is constant, the area A must grow with n. Given two nodes separated by distance d, the number of hops needed to send a packet from one node to the other is given by $u = \frac{d}{z}$, where z is the average forward progress made toward a destination in the course of one transmission.

The protocols are compared under uniform and localized query access patterns. By uniform access patterns, we mean that queries are uniformly distributed over all data items in the database. Formally: $\lambda_{ij} = \frac{\lambda_i}{n}$. In the localized access patterns, nodes are more likely to access data items that are close-by than those that are far away. If we consider that H_i is the number of hops between node i and the farthest node in the network, we assume that i generate queries to the nodes that are l hops away from it with mean rate of λ_i^l. Formally:

$$\lambda_i^l = \begin{cases} \dfrac{\lambda_i}{2^l} & \text{if} \quad 1 \leq l \leq H_i - 1 \\ \lambda_i - \displaystyle\sum_{j=1}^{H_i-1} \lambda_i^j & \text{if} \quad l = H_i \end{cases}$$

For a network of size A, the probability density function (pdf) of the distance S

between two nodes with Random Waypoint (RWP) mobility model is given by [18]:

$$f_S(s) = \frac{\widehat{s}}{9\pi}[(-36\widehat{s}^2 + 24)\pi + (72\widehat{s}^2 - 48)\arcsin(\frac{\widehat{s}}{2}) + (-\widehat{s}^5 + 16\widehat{s}^3 + 12\widehat{s}\sqrt{4 - \widehat{s}^2})]$$

We assume that all nodes have the same transmission radio range. Two nodes establish a link if they are located within distance of r of each other. The probability that the distance between two nodes is less than or equal to r is given by [47]:

$$p_m = P(X \leq r) = \int_0^r f_S(s)ds$$

For a node with a battery power capacity of C Joules, the probability p_a that a node is alive is given in Chapter 6:

$$p_a = \sum_{i=1}^{C} \Pi_i = 1 - \Pi_0 = \frac{\frac{\lambda}{\mu} - (\frac{\lambda}{\mu})^{C+1}}{1 - (\frac{\lambda}{\mu})^{C+1}}$$

such as: λ and μ denote the rate at which the residual battery power R increases and decreases respectively.

The probability of establishing a link (g), and the probability of establishing a path of i hops (P_i)are also given in Chapter 6. The probability that two nodes are connected is then obtained as $Pc = \sum_{i=1}^{N-1} P_i$.

7.4.2 Localized K-hop cluster-based Data replication scheme

Replication cost

For the sake of simplicity, we assume that a stable K-hop cluster is a square with $(2K \times z)$ hops on each side, and the group leader is in the center of this square. The replication cost is $\frac{A}{4K^2z^2}$. As z is a constant and n is proportional to A, c_r is a function of $O(\frac{n}{K^2})$.

Storage cost

The storage cost $c_s = (\frac{n}{c_r})$. Thus, $c_s = O(K^2)$.

Update cost

When a node i updates its D_i, it sends an update packet to its group leader, which costs $O(K)$ packets. Moreover, each group leader GL_i sends an aggregate update message at a rate of $\overline{\mu_{GL_i}} = \frac{1}{\Delta T_{GL_i}}$ per second. If we consider that there are at most $2K$ hops between adjacent group leaders. Therefore, the update cost is:

$$\begin{aligned} c_u &= (\tfrac{1}{\mu_i}K) + (\tfrac{1}{\Delta T_{GL_i}})2K(\tfrac{n}{4K^2}) \\ &= ((\frac{1}{\mu_i}K) + (\overline{\mu_{GL_i}}\frac{n}{2K})) \\ &= O(K + \frac{n}{K}) \text{ packets/sec} \end{aligned}$$

Query cost

Each query packet needs at most K hops to reach the group leader. Thus, $c_q = O(K)$ packets/sec.

Availability degree

In our replication scheme, when a node knows that it is about to be disconnected from its server, it download all the data items that will be unreachable. So, all the queries that generated for these data items will success even if the nodes holding those data item become unreachable.

Under uniform access patterns, there are $O(K^2)$ queries generated for data items belonging to the group. So, the availability degree can be written as:

$$\begin{aligned} Ad_i &= \frac{1}{\lambda_i}((\frac{\lambda_i}{n})K^2 + (n - K^2)\frac{\lambda_i}{n}P_c) \\ &= P_c + (1 - P_c)\frac{K^2}{n} \end{aligned}$$

Under localized access patterns, there are $(\sum_{l=1}^{K} \lambda_i^l)$ queries generated for data items belonging to the group. So, the availability degree can be written as:

$$\begin{aligned} Ad_i &= \frac{1}{\lambda_i}(\sum_{l=1}^{K} \lambda_i^l + P_c \sum_{l=K+1}^{H} \lambda_i^l) \\ &= P_c + (1 - \frac{1}{2^K})(1 - P_c)) \end{aligned}$$

Accuracy degree

Consider the time interval t between the current propagation of D_x and the immediately preceding propagation of D_x by the group leader lid_x.

A query operation made by node y for a specific data item D_x succeeds if it returns the most recent value of D_x. The query operation may succeed if at least one of the following two cases occur:

- *Case 1:* The data item D_x has not been updated during the time interval t (see figure 7.3.(a)).
- *Case 2:* During the time interval t, the first update for the data item D_x occurred at time t'. Between t' and the time of the current update propagation, D_x is updated at least once and node y does not perform any query operation (see figure 7.3.(b)).

The probabilities that Case 1 and Case 2 occur are computed as follows:

$$\begin{aligned} P[Case\ 1\ occurs] &= \int_0^{+\infty} \exp^{-\overline{\mu_{GL_x}}t} \exp^{-\mu_x t}(1-\exp^{-\lambda_{yx}t})dt \\ &= \frac{\lambda_{yx}}{(\mu_x+\overline{\mu_{GL_x}})(\lambda_{yx}+\mu_x+\overline{\mu_{GL_x}})} \end{aligned}$$

$$\begin{aligned} P[Case\ 2\ occurs] &= P[no\ update\ during\ t']P[no\ query\ during\ t-t'] \\ &= \int_0^{+\infty} \exp^{-\overline{\mu_{GL_x}}t}(\int_0^t \lambda_{yx} \exp^{-\overline{\mu_x}t} \exp^{-\lambda_{yx}t}\, dt')dt \\ &= \frac{\lambda_{yx}}{(\lambda_{yx}+\overline{\mu_{GL_x}})(\mu_x+\overline{\mu_{GL_x}})} \end{aligned}$$

The probability that node y obtain the most recent value of D_x is the sum of the probabilities for Case 1 and Case 2 and is given by: $Ac = Ad(P[Case1] + P[Case2])$.

7.4.3 Derhab's protocol [36]

In Derhab's protocol [36], there is only one server in each network partition. Each server stores the whole database. The protocol provide full data availability but it does not consider data update.

7.4.4 RRs protocol

In [125], a few elected nodes inside each region are servers. At worst case, it can lead to the complexity of $O(n)$ servers. The other scalability metrics are proportional to $O(\sqrt{n})$. As a node sends its query to a single region, so the probability that the query packet can reach the server of the region is P_c.

7.4.5 CDRA protocol

In CDRA [154], a query packet may be propagated in the whole network, which leads to the complexity of $O(n)$. The update operation is performed in two phases, which means a complexity of $O(2n)$. In CDRAR, the node which requested the data item is chosen to be a replica server. So, this solution does not consider the replication of non-requesting data items, which makes the availability degree of CDRA depends on the value of P_c.

7.4.6 Discussion

Table 7.1: Comparison of data replication protocols

	Our scheme	Derhab	RRs	CDRA
Replication cost	$O(\frac{n}{K^2})$	*number of partitions*	$O(n)$	$O(n)$
Update cost	$O(K + \frac{n}{K})$	N/A	$O(\sqrt{n})$	$O(2n)$
Query cost	$O(K)$	$O(\sqrt{n})$	$O(\sqrt{n})$	$O(n)$
Storage cost	$O(K^2)$	$O(n)$	$O(\sqrt{n})$	$O(1)$
Availability	$P_c + (1 - P_c)\frac{K^2}{n}$ (uniform) $P_c + (1 - \frac{1}{2^K})(1 - P_c)$ (localized)	1 (both)	P_c (both)	P_c (both)
Accuracy	Ad(P[Case1]+P[Case 2])	N/A	P_c	P_c

Table 7.2: Performance of the proposed replication scheme versus K

	$K = O(1)$	$K = O(\sqrt[4]{n})$	$K = O(\sqrt{n})$
Replication cost	$O(n)$	$O(\sqrt{n})$	$O(1)$
Update cost	$O(n)$	$O(n)^{\frac{3}{4}}$	$O(\sqrt{n})$
Query cost	$O(1)$	$O(\sqrt[4]{n})$	$O(\sqrt{n})$
Storage cost	$O(1)$	$O(\sqrt{n})$	$O(n)$
Availability	$P_c + (1 - P_c)\frac{1}{n}$ (uniform) $\frac{1}{2}(1 + P_c)$ (localized)	$P_c + (1 - P_c)\frac{1}{\sqrt{n}}$ (uniform) $P_c + (1 - \frac{1}{2\sqrt[4]{n}})(1 - P_c)$ (localized)	1 (uniform) 1 (localized)
Accuracy	Ad(P[Case1]+P[Case 2])	Ad(P[Case1]+P[Case 2])	Ad(P[Case1]+P[Case 2])

We summarize the performance metrics of the four replication schemes in Table 7.1. In the table, the values for these metrics represent worst case behavior. CDRA shows poor scalability, since it floods the whole network to locate the target node, which requires a cost of $O(n)$. Our scheme gives higher data availability degree than RRs and CDRA. It outperforms the other replication protocols in terms of query cost. Our scheme loses some accuracy in the favor of update cost. Minimizing the update cost of our scheme with respect to K, we get $K = O(\sqrt{n})$, and $c_u = O(\sqrt{n})$. The variable K is upper-bounded by H, which is the longest path in the network. H is known to be proportional to $O(\sqrt{n})$ [95]. When $K = H$, our scheme has an asymptotic update and query cost of $O(\sqrt{n})$. The performance results of our scheme depend on K. So to make a fair comparison, we calculate the performance metrics of our scheme when $K = O(1)$, $K = O(\sqrt[4]{n})$, and $K = O(\sqrt{n})$, as shown in Table 7.2. In the table, we can notice that the scalability costs of our scheme is slightly similar to to CDRA when $K = O(1)$ except in the case of query cost. When $K = O(\sqrt{n})$, our scheme provides results that are slightly similar to Derhab's protocol. It can provide full data availability since it considers the whole network as one group. For all the values of K, our scheme still outperforms RRs and CDRA in terms of data availability. We can also notice that when K increases, our scheme becomes more optimal in terms of replication cost, update cost, availability degree, and accuracy degree. It is also obvious that the query cost and the storage cost of our scheme are more optimal when K is low.

7.5 Conclusion

In this chapter, we have presented a localized hybrid data delivery protocol that combines the push-based and the pull-based data delivery approaches. The protocol divides the network into disjoint stable K-hop DAGs. We have used three levels of local knowledge: First, the 1-hop knowledge, that allows to estimate the residual time of wireless links and to decide whether a stable K-hop DAG will partition or not. Second, the intra-group information (i.e., K-hop knowledge) that permits (1) nodes to send their queries and updates to nearby database servers, and (2) helps group leaders to estimate the update propagation period. Third, the inter-group information reduce the cost of update propagation since group leaders send the update message only to their adjacent group leaders. We have also proposed a partition prediction

algorithm that can tolerate concurrent topology changes. We have analyzed the scalability, the availability, and the accuracy of our replication scheme as well as four other protocols under different query access patterns. Analysis has shown that our scheme is asymptotically better than Derhab's protocol, RRs, and CDRA in terms of query cost. In addition, its availability degree is higher than that of RRs and CDRA for all the values of K. It also operates as CDRA when $K = O(1)$ and as Dehab's protocol when $K = O(\sqrt{n})$. The data replication scheme presented in this chapter looks promising in terms of the results presented, and in its ability to offer a good trade-off between availability, accuracy, and scalability.

General Conclusion

This thesis has addressed the challenging and fundamental issues related to designing a replication protocol in the field of mobile ad-hoc networks, i.e. issues that are specific for ad hoc networks, and which significantly degrade the performance of data access. These issues are: network partitioning, energy consumption, and scalability. We have classified the existing replication protocols compared based on how they addressed these issues.

While current solutions are known to be either partition-aware or scalable, there has no considerable effort in the research of replication protocols that address more than one issue. None of the existing data replication techniques addresses all the MANET issues, that is, none of them is energy-aware, partition-aware as well as scalable. Our contribution is the proposition of six localized data replication protocols that address at least two issues. The goals of these protocols is to achieve higher data availability with lower cost.

We have firstly proposed an algorithm that can predict network partitioning before its occurrence. This algorithm is utilized by a replication protocol that replicates a centralized database in the future separate network partitions. Simulation results has shown that the protocol achieves high data availability (i.e. full service coverage) with a low replication cost. However, the proposed algorithm is inefficient in ad hoc mobile networks because they converge to a stable state only if the topological changes stop. When the system experiences a new topological change before completing the convergence, the algorithm restarts convergence to its legitimate state from scratch. To fix this shortcoming, a self-stabilizing partition prediction algorithm that can within a finite time converge to a stable state even if topological changes occur during the convergence time.

We have also proposed three location-based data replication protocols, which are: FDRS, HDRS, and FSLS. They operate on a position-based routing protocol that

needs only to know the positions of its neighbor nodes to make a forwarding decision, and it does not need need to keep global states for routing data packets. These architecture helps the replication protocol to be more scalable to large ad-hoc networks. The location service FSLS can offer a good trade-off between availability and scalability. We have shown that FSLS scales well with increasing network size. Its overhead cost can be proportional to $(v\sqrt[4]{n})$, especially under localized traffic. Analysis and simulation results show that FSLS comes second after CRLS [130] in terms of location availability, and it is the closest competitor to HIGH-GRADE [151] in terms of scalability. FDRS and HDRS are asymptotically better than the Rendezvous Regions (RRs) protocols [125] in terms of scalability and availability.

The sixth proposed protocol, is a localized cluster-based data replication protocol, it addresses all the issues identified in the thesis. It can make a tradeoff between availability, consistency and update cost. It can operate as a cluster-based protocol [154] when $K = O(1)$, and as a partition-aware replication protocol [36] when $K = O(\sqrt{n})$.

One avenue of future work is to implement a scalable partition prediction algorithm that can achieve both hight data availability and high scalability. We also aim to study how far data availability is affected in situations where nodes crash, or when are attacks are launched against nodes, such as: a denial of service (DOS) attack that disables the service provided by the critical node or the sleep deprivation torture attack that exhausts the battery of its victim nodes. Based on this study, we will propose a replication protocol that determines the appropriate nodes that can host replicas of data items.

Bibliography

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[3] James F. Allen and George Ferguson. Actions and events in interval temporal logic. Technical report TR521, University of Rochester, Rochester, NY, USA, 1994.

[4] Alan D. Amis, Ravi Prakash, Thai H.P. Vuong, and Dung T. Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *Proceedings of IEEE INFOCOM 2000*, pages 32–41, 2000.

[5] S. apkun, M. Hamdi, and J. Hubaux. Gps-free positioning in mobile ad-hoc networks. In *Proceedings of the 34th Hawaii International Conference on System Sciences*, 2001.

[6] S. apkun, M. Hamdi, and J. P. Hubaux. Gps-free positioning in mobile ad-hoc networks. *Cluster Computing Journal*, 5(2), April 2002.

[7] N. Asokan and Philip Ginzboorg. Key agreement in ad hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.

[8] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.

[9] I. Aydin and C.-C. Shen. Facilitating match-making service in ad hoc and sensor networks using pseudo quorum. In *Proceedings of the 11th IEEE International*

Conference on Computer Communications and Networks (ICCCN), pages 4–9, 2002.

[10] H. Baala, O. Flauzac, J. Gaber, M. Bui, and T. El-Ghazawi. A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks. *Journal of Parallel and Distributed Computing*, 63(1):97–104, 2003.

[11] O. Babaoglu, R. Davoli, A. Montresor, and R. Segala. System support for partition-aware network applications. In *Proceedings of the The 18th International Conference on Distributed Computing Systems (ICDCS '98)*, page 184, Washington, DC, USA, 1998. IEEE Computer Society.

[12] Nadjib Badache, Djamel Djenouri, and Abdelouahid Derhab. Mobility impact on mobile ad hoc routing protocols. In *ACS/IEEE International Conference on Computer System and Applications (AICCSA'03)*, 2003.

[13] Nadjib Badache, Djamel Djenouri, Abdelouahid Derhab, and Tayeb Lemlouma. Les protocoles de routage dans les réseaux mobiles ad hoc. *Revue d'Information Scientifique et Technique (RIST)*, 12(2):77–112, Septembre 2003.

[14] Valmir C. Barbosa. *An introduction to distributed algorithms.* MIT Press, Cambridge, MA, USA, 1996.

[15] S. Basagni, I. Chlamtac, V.R. Syrotiuk, and B.A. Woodward. A distance routing effect algorithm for mobility (dream). In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 76–84, 1998.

[16] Stefano Basagni, Marco Conti, Silvia Giordano, and Ivan Stojmenovic. *Mobile Ad Hoc Networking.* Wiley-IEEE Press, 2004.

[17] Gunter Belch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications.* John Wiley & Sons, Inc., 1998.

[18] Christian Bettstetter. Topology properties of ad hoc networks with random waypoint mobility. In *MOBIHOC03*, June 2003.

[19] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. Macaw: A media access protocol for wireless lans. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM '94)*, page 212225, 1994.

[20] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *3rd international Workshop on Discrete Algorithms and methods for mobile computing and communications*, 1999.

[21] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, 1998.

[22] N. Bulusu, J. Heidemann, and D. Estrin. Gps-less low-cost outdoor localization for very small devices. *IEEE Personal Communications*, 7(5):28–34, October 2000.

[23] Tracy Camp, Je. Boleng, Brad Williams, Lucas Wilcox, and William Navidi. Performance comparison of two location based routing protocols for ad hoc networks. In *INFOCOM 2002*, 2002.

[24] Guohong Cao, Liangzhong Yin, and Chita R. Das. Cooperative cache-based data access in ad hoc networks. *Computer*, 37(2):32–39, 2004.

[25] K. Chen and K. Nahrstedt. An integrated data lookup and replication scheme in mobile ad hoc networks. In *Proc. of SPIE International Symposium on the Convergence of Information Technologies and Communications (ITCom 2001)*, August 2001.

[26] K. Chen, S.H. Shah, and K. Nahrstedt. Cross-layer design for data accessibility in mobile ad hoc networks. In *Proc. of 5th World multiconference on systemics, cybernetics and informatics (SCI 2001)*. Orlando, Florida, July 2001.

[27] Y. Chen and J. L. Welch. Self-stabilizing mutual exclusion using tokens in mobile ad hoc networks. In *the 6th Annual International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM'2002)*, 2002.

[28] C. T. Cheng, H. L. Lemberg, S. J. Philip, E. van den Berg, and T. Zhang. Slalom: A scalable location management scheme for large mobile ad-hoc networks. In *Proceedings of IEEE Wireless Communications and Networking Conference*, March 2002.

[29] Thomas Clausen. Comparative study of routing protocols for mobile ad-hoc networks. technical report 5135, INRIA, France, Mars 2004.

[30] George F. Coulouris and Jean Dollimore. *Distributed systems: concepts and design*. Addison-Wesley Longman Publishing Co., Inc., 3rd edition, 1988.

[31] S. R. Das, R. Castaneda, and J. Yan. Simulation based performance evaluation of mobile, ad hoc network routing protocols. *ACM/Baltzer Mobile Networks and Applications (MONET) Journal*, pages 179–189, July 2000.

[32] Saumitra M. Das, Himabindu Pucha, and Charlie Hu. Performance comparison of scalable location services for geographic ad hoc routing. In *Proceedings of IEEE INFOCOM 2005*, March 2005.

[33] S.K. Das, A. Mukherjee, S. Bandyopadhyay, D. Saha, and K. Paul. An adaptive framework for qos routing through multiple paths in ad−hoc wireless networks. *Journal of Parallel and Distributed Computing*, 63:141–153, 2003.

[34] J. Deng and Z. J. Hass. Dual busy tone multiple access (dbtma): A new medium access control for packet radio networks. In *Proceedings of the IEEE International Conference on Universal Personal Communications*, pages 314–318, 1998.

[35] Abdelouahid Derhab and Nadjib Badache. Localized hybrid data delivery scheme using k-hop clustering algorithm in ad hoc networks. In *Proceedings of the 3rd IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, (MASS 2006)*, October 2006.

[36] Abdelouahid Derhab and Nadjib Badache. A pull-based service replication protocol in mobile ad hoc networks. *European Transactions on Telecommunications*, 18(1):1–11, 2007.

[37] Abdelouahid Derhab and Nadjib Badache. A distributed mutual exclusion algorithm over multi-routing protocol for mobile ad hoc networks. *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, To appear.

[38] Abdelouahid Derhab and Nadjib Badache. Balancing the tradeoffs between scalability and availability in mobile ad-hoc networks with a flat hashing-based location service. *ELSEVIER Ad Hoc Networks Journal*, To appear, available online 12 October 2007.

[39] Abdelouahid Derhab and Nadjib Badache. Self-stabilizing leader election algorithm in highly dynamic ad hoc mobile networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, To appear, available online 22 October 2007.

[40] Abdelouahid Derhab and Nadjib Badache. Self-stabilizing algorithm for high service availability in spite of concurrent topology changes in ad hoc mobile networks. *Journal of Parallel and Distributed Computing (JPDC)*, Under revision.

[41] Abdelouahid Derhab, Nadjib Badache, and Abdelmadjid Bouabdallah. A partition prediction algorithm for service replication in mobile ad hoc networks. In *2nd International Conference on Wireless on Demand Network Systems and Service (WONS 2005)*, pages 236–245, January 2005.

[42] Abdelouahid Derhab, Nadjib Badache, Karim Tari, and Sihem Sami. Els: Energy-aware some-for-some location service for ad hoc mobile networks. In *Proceedings of International Conference on Wireless Algorithms, Systems, and Applications (WASA 2006)*, pages 240–251, August 2006.

[43] Djamel Djenouri, Abdelouahid Derhab, and Nadjib Badache. Ad hoc networks routing protocols and mobility. *International Arab Journal for Information Technology*, 3(2):126–133, April 2006.

[44] Shlomi Dolev, Elad Schiller, and Jennifer Welch. Random walk for self-stabilizing group communication in ad hoc networks. In *Proceedings of the 21th*

annual symposium on Principles of distributed computing (PODC'02), pages 259–259, 2002.

[45] S. R. Dunbar. The average distance between points in geometric figures. *College Mathematics Journal*, 28(3):187–197, May 1997.

[46] M. Duque-Anton, F. Bruyaux, and P. Semal. Measuring the survivability of a network: Connectivity and rest-connectivity. *European Transactions on Telecommunications*, 11(2):149–159, 2000.

[47] Golnaz Farhadi and Norman C. Beaulieu. On the connectivity and average delay of mobile ad hoc networks. In *IEEE International Conference on Communications (ICC 2006)*, 2006.

[48] Yaacov Fernandess and Dahlia Malkhi. K-clustering in wireless ad hoc networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing (POMC '02)*, pages 31–37, 2002.

[49] Chane L. Fullmer and J. J. Garcia-Luna-Aceves. Floor acquisition multiple access (fama) for packet-radio networks. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM '95)*, pages 262–273, 1995.

[50] M. Gerla and J. T.-C. Tsai. Multicluster, mobile, multimedia radio network. *ACM/Baltzer Wireless Networks*, 1(3):255–265, 1995.

[51] T. Goff, N. B. Abu-Ghazaleh, D. S. Phatak, and R. Kahvecioglu. Preemptive routing in ad hoc networks. In *Proceedings of the 7th annual international conference on Mobile computing and networking (MobiCom '01)*, pages 43–52, 2001.

[52] D. Goyal and J. Caffery. Partitioning avoidance in mobile ad hoc networks using network survivability concepts. In *Proceedings of the 7th IEEE Symposium on Computers and Communications (ISCC)*, pages 553–558, Taormina, Italy, July 2002.

[53] Sandeep K. S. Gupta and Pradip K. Srimani. Self-stabilizing multicast protocols for ad hoc networks. *Journal of Parallel and Distributed Computing*, 63(1):87–96, 2003.

[54] Z. Haas and B. Liang. Ad hoc mobility management with randomized database groups. In *Proceedings of IEEE ICC*, June 1999.

[55] Zygmunt J. Haas and Ben Liang. Ad hoc mobility management with uniform quorum systems. *IEEE/ACM Transactions on Networking*, 7(2):228–240, 1999.

[56] Zygmunt J. Haas and Marc R. Pearlman. The performance of query control schemes for the zone routing protocol. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 167–177, 1998.

[57] T. Hara. Effective replica allocation in ad hoc networks for improving data accessibility. In *Proc. of IEEE Infocom 2001*, volume 3, pages 1568–1576, April 2001.

[58] Takahiro Hara. Replica allocation methods in ad hoc networks with data update. *Mobile Networks and Applications*, 8(4):343–354, 2003.

[59] Takahiro Hara. Location management of data items in mobile ad hoc networks. In *Proceedings of the 2005 ACM symposium on Applied computing (SAC '05)*, pages 1174–1175, 2005.

[60] Takahiro Hara. Location management of replicas considering data update in ad hoc networks. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06)*, pages 753–758, 2006.

[61] Takahiro Hara, Yin-Huei Loh, and Shojiro Nishio. Data replication methods based on the stability of radio links in ad hoc networks. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA '03)*, pages 969–973, 2003.

[62] Z. J. Hass and J. Deng. Dual busy tone multiple access (dbtma): Performance evaluation. In *49th Annual International Vehicular Technology Conference*, October 1998.

[63] Kostas P. Hatzis, George P. Pentaris, Paul G. Spirakis, Vasilis T. Tampakas, and Richard B. Tan. Fundamental control algorithms in mobile networks. In *Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures (SPAA '99)*, pages 251–260, 1999.

[64] M. Hauspie, A. Panier, and D. Simplot-Ryl. Localized probabilistic and dominating set based algorithm for efficient information dissemination in ad hoc networks. In *IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2004)*, October 2004.

[65] Michaël Hauspie, David Simplot, and Jean Carle. Replication decision algorithm based on link evaluation for services in manet. Technical Report 2002-05, IRCICA/LIFL, Univ. Lille 1, 2002.

[66] Michaël Hauspie, David Simplot, and Jean Carle. Partition detection in mobile ad−hoc networks. In *Proceedings of the 2nd Mediterranean Workshop on Ad−Hoc Networks*. Mahdia, Tunisia, June 2003.

[67] Hideki Hayashi, Takahiro Hara, and Shojiro Nishio. Updated data dissemination methods for updating old replicas in ad hoc networks. *Personal and Ubiquitous Computing*, 9(5):273–283, 2005.

[68] Abdelsalam A. Helal, Bharat K. Bhargava, and Abdelsalam A. Heddaya. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, 1996.

[69] T.-C. Hou and T.-J. Tsai. An access-based clustering protocol for multihop wireless ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 19(7), July 2001.

[70] Yih-Chun Hu, David B. Johnson, and Adrian Perrig. Secure ecient distance vector routing in mobilewireless ad hoc networks. In *Fourth IEEEWorkshop on Mobile Computing Systems and Applications WMCSA 02*, June 2002.

[71] Jiun-Long Huang, Ming-Syan Chen, and Wen-Chih Peng. Exploring group mobility for replica data allocation in a mobile environment. In *Proceedings of the twelfth international conference on Information and knowledge management (CIKM '03)*, pages 161–168, 2003.

[72] J. Hubaux, L. Buttyan, and S. Capkun. The quest for security in mobile ad hoc networks. In *Proceeding of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC'99)*, 1999.

[73] R. Jain, A. Puri, and R. Sengupta. Geographical routing using partial information for wireless ad hoc networks. *IEEE Personal Communications*, pages 48–57, February 2001.

[74] D. B. Jhonson and D. A. Maltz. Dynamic source routing in ad hoc wireless. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[75] Weijia Jia and Wanlei Zhou. *Distributed Network Systems: From Concepts to Implementations*. Springer-Verlag New York, Inc., 2006.

[76] C. Jones and M. Matari. Automatic synthesis of communication-based coordinated multi-robot systems. Technical Report CRES-04-007, University of Southern California Center for Robotics and Embedded Systems, 2004.

[77] M. Jorgic, I. Stojmenovic, M. Hauspie, and D. Simplot-Ryl. Localized algorithms for detection of critical nodes and links for connectivity in ad hoc networks. In *Proceedings of the 3^{rd} Annual Mediterranean Ad Hoc Networking Workshop Med-Hoc-Net*, pages 360–371, Bodrum, Turkey, June 2004.

[78] Hirotsugu Kakugawa and Masafumi Yamashita. A dynamic reconfiguration tolerant self-stabilizing token circulation algorithm in ad-hoc networks. In *Principles of Distributed Systems, 8th International Conference (OPODIS 2004)*, pages 256–266, 2004.

[79] E. Kaplan. *Understanding GPS: Principles And Applications*. Artech House, 1996.

[80] P. Karn. Macaa new channel access method for packet radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, page 134140, 1990.

[81] B. Karp and H. T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 243–254, 2000.

[82] Goutham Karumanchi, Srinivasan Muralidharan, and Ravi Prakash. Information dissemination in partitionable mobile ad hoc networks. In *Symposium on Reliable Distributed Systems*, pages 4–13, 1999.

[83] Leonard Kleinrock. *Queueing Systems, Volume 1:Theory.* Wiley-Interscience, 1975.

[84] Y. Ko and N.H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 66–75, 1998.

[85] Jiejun Kong, Petros Zerfos, Haiyun Luo, Songwu Lu, and Lixia Zhang. Providing robust and ubiquitous security support for mobile ad-hoc networks. In *International Conference on Network Protocols (ICNP)*, pages 251–260, 2001.

[86] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *Proceedings of 17th IEEE Symposium Logic in Computer Science (LICS'2002)*, pages 383–392. Copenhagen, Denmark, IEEE Computer Society Press, July 2002.

[87] Hyunyoung Lee, Jennifer L. Welch, and Nitin H. Vaidya. Location tracking with quorums in mobile ad hoc networks. *Ad Hoc Networks, Elsevier Science*, 1(4):371–381, November 2003.

[88] S. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In *Proceedings of the IEEE International Conference on Communications*, pages 3201–3205, 2001.

[89] Brian Lehane, Linda Dolye, and Donal O'Mahony. Ad hoc key management infrastructure. In *ITCC (2)*, pages 540–545, 2005.

[90] R. Leung, J. Liu, E. Poon, C. Chang, and B. Li. Mp−dsr: A qos−aware multi−path dynamic source routing protocol for wireless networks. In *Proceedings of the 26th IEEE Annual Conference on Local Computer Networks (LCN 2001)*. Tampa, Florida, November 2001.

[91] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 120–130, 2000.

[92] Ben Liang and Zygmunt J. Haas. Virtual backbone generation and maintenance in ad hoc network mobility management. In *INFOCOM (3)*, pages 1293–1302, 2000.

[93] W.-H. Liao, Y.-C. Tseng, and J.-P. Sheu. Grid: a fully location-aware routing protocol for mobile ad hoc networks. *Telecommunication Systems*, 18:6184, 2001.

[94] C. R. Lin and M. Gerla. Maca/pr: An asynchronous multimedia multihop wireless network. In *Proceedings of IEEE INFOCOM 97*, April 1997.

[95] X. Lin and I. Stojmenovic. Location− based localized alternate, disjoint and multi−path routing algorithms for wireless networks. *Journal of Parallel and Distributed Computing*, 2002.

[96] J. Luo, J.-P. Hubaux, and P.Th. Eugster. PAN: Providing reliable storage in mobile ad hoc networks with probabilistic quorum systems. In *Proc. of the 4nd ACM/SIGMOBILE Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03)*, pages 1–12, 2003.

[97] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

[98] M. Maleki, K. Dantu, and M. Pedram. Lifetime prediction routing in mobile ad hoc networks. In *Proc of IEEE Wireless Communication and Networking Conference (WCNC'03)*, March 2003.

[99] Dahlia Malkhi, Michael K. Reiter, Avishai Wool, and Rebecca N. Wright. Probabilistic quorum systems. *Information and Computation*, 170(2):184–206, 2001.

[100] N. Malpani, J. L. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 96–103, 2000.

[101] Toshimitsu Masuzawa and Hirotsugu Kakugawa. Self-stabilization in spite of frequent changes of networks: Case study of mutual exclusion on dynamic rings. In *Self-Stabilizing Systems, 7th International Symposium, SSS 2005*, pages 183–197, 2005.

[102] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network*, 15(6):30–39, November/December 2001.

[103] A. B. McDonald and T. F. Znati. A mobility-based framework for adaptive clustering in wireless ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1466–1487, August 1999.

[104] B. Narendran, P. Agrawal, and D. K. Anvekar. Minimizing cellular handover failures without channel utilization loss. In *Proceedings of IEEE Global Communications Conference*, pages 1679–1685, San Francisco, CA, December 1994.

[105] B. Nath and D. Niculescu. Routing on a curve. *ACM SIGCOMM Computer Communication Review*, 33(1):155–160, 2003.

[106] Sanket Nesargi and Ravi Prakash. MANETconf: Configuration of hosts in a mobile ad hoc network. In *Proceeding of the IEEE INFOCOM 2002*, pages 1059–1068, 2002.

[107] Dan Nguyen, Li Zhao, Pra ornsiri Uisawang, and John Platt. Security routing analysis for mobile ad hoc networks. Technical report, University of Colorado, 2003.

[108] Pavan Nuggehalli, Vikram Srinivasan, and Carla-Fabiana Chiasserini. Energy-efficient caching strategies in ad hoc wireless networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '03)*, pages 25–34, 2003.

[109] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of IEEE INFOCOM 97*, pages 1405–1413, April 1997.

[110] Pradeep Parvathipuram, Vijay Kumar, and Gi-Chul Yang. An efficient leader election algorithm for mobile ad hoc networks. In *Proceedings of the First International Conference on Distributed Computing and Internet Technology (ICDCIT)*, pages 32–41, 2004.

[111] Marc R. Pearlman, Jing Dengy, Ben Liangz, and Zygmunt J. Haasx. Elective participation in ad hoc networks based on energy consumption. In *IEEE GLOBECOM 2002*, pages 17–21. Taipei, Taiwan, November 2002.

[112] G. Pei and M. Gerla. Mobility management in hierarchical multi-hop mobile wireless networks. In *Proceedings of IEEE ICCCN'99*, pages 324–329. Boston, MA, October 1999.

[113] C. Perkins, E. Royer, and S. Das. Ip address autoconfiguration for ad hoc networks. Internet-draft, IETF, 2000.

[114] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computer. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[115] C. E. Perkins and E. M.Royer. Ad hoc on demand distance vector (aodv) algorithm. In *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, pages 90–100, February 1999.

[116] Charles E. Perkins. *Ad Hoc Networking*. Addison Wesly Longman Publishing Co., Inc., 2001.

[117] Sumesh J. Philip, Joy Ghosh, and C. Qiao. Performance evaluation of a multi-level hierarchical location management protocol for ad hoc net works. *Elsevier Computer Communications, Special issue on Performance Issues of Wireless LANs, PANs, and Ad Hoc Networks*, 8(10):1110–1122, 2005.

[118] L. Qin and T. Kunz. Pro-active route maintenance in dsr. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(3):79–89, 2002.

[119] L. Qin and T. Kunz. Increasing packet delivery ratio in dsr by link prediction. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, pages 300–309, Hawaii, USA, January 2003. IEEE Computer Society Press.

[120] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. Ght: A geographic hash table for data-centric storage in sensornets. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[121] D. Reichardt, M. Miglietta, L. Moretti, P. Morsink, and W. Schulz. Cartalk 2000 - safe and comfortable driving based upon inter-vehicle-communication. In *Proceedings of IEEE Intelligent Vehicle Symposium (IV'02)*, 2002.

[122] C. L. Richard and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, September 1997.

[123] Gruia-Catalin Roman, Qingfeng Huang, and Ali Hazemi. Consistent group membership in ad hoc networks. In *International Conference on Software Engineering*, pages 381–388, 2001.

[124] Gruia-Catalin Roman, Qingfeng Huang, and Ali Hazemi. Consistent group membership in ad hoc networks. In *International Conference on Software Engineering*, pages 381–388, 2001.

[125] Karim Seada and Ahmed Helmy. Rendezvous regions: A scalable architecture for service location and data-centric storage in large-scale wireless networks. In *IPDPS*, 2004.

[126] Boon-Chong Seet, Yan Pan, Wen-Jing Hsu, and Chiew-Tong Lau. Multi-home region location service for wireless ad hoc networks: An adaptive demand-driven approach. In *Proceedings of the 2nd Annual Conference on Wireless On-demand Network Systems and Services (WONS'05)*, pages 258–263, 2005.

[127] Samba Sesay, Zongkai Yang, Biao Qi, and Jianhua He. Simulation comparison of four wireless ad hoc routing protocols. *Information Technology Journal*, 3(3):219–226, 2004.

[128] J. P. Singh, A. Ahuja, and S. Agarwal. Link connectivity assessment based applications for mobile ad-hoc networks. In *42nd Annual Technical Convention of The Institution of Electronics and Telecommunication Engineers*. New Delhi, India, September 2000.

[129] J. So and N. Vaidya. Mtsf: A timing synchronization protocol to support synchronous operations in multihop wireless networks. Technical report, University of Illinois, Urbana-Champaign, October 2004.

[130] I. Stojmenovic. A scalable quorum based location update scheme for routing in ad hoc wireless networks. Technical Report TR-99-09, University of Ottawa, September 1999.

[131] I. Stojmenovic and X. Lin. Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless network. *IEEE Transactions on Parallel and Distributed Systems*, 12, 2001.

[132] W. Su, S-J. Lee, and M. Gerla. Mobility prediction and routing in ad hoc wireless networks. *International Journal of Network Management*, 11(3):30, 2001.

[133] F. Talucci and M. Gerla. Maca-bi (maca by invitation) a wireless mac protocol for high speed ad hoc networking. In *Proceedings of ICUPC97*, November 1997.

[134] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal Computing*, 1(2):149–160, June 1972.

[135] Jivodar B. Tchakarov and Nitin H. Vaidya. Efficient content location in wireless ad hoc networks. In *Mobile Data Management (MDM'04)*, 2004.

[136] Vineet Thanedar, Kevin C. Almeroth, and Elizabeth M. Belding-Royer. A lightweight content replication scheme for mobile ad hoc environments. In *NETWORKING*, pages 125–136, 2004.

[137] Y.−C. Tseng, S.−Y. Ni, and Y.−S. Chen. The broadcast storm problem in a mobile ad hoc network. *ACM Wireless Networks*, 8(2):152–167, March 2002.

[138] N. Vaidya. Weak duplicate address detection in mobile ad hoc networks. In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, june 2002.

[139] Sudarshan Vasudevan, Jim Kurose, and Don Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *12th IEEE International Conference on Network Protocols (ICNP 2004)*, pages 350–360, Berlin, Germany, October 2004.

[140] J. Walter, G. Cao, and M. Mohanty. A k-mutual exclusion algorithm for wireless ad hoc networks. In *Proc of the first annual Workshop on Principles of Mobile Computing (POMC 2001)*, Newport, Rhode Island USA, August 2001.

[141] J. Walter, J. Welch, and N. Vaidya. A mutual exclusion algorithm for ad hoc mobile networks. In *Dial M for Mobility workshop*, Dallas, TX, USA, October 1998.

[142] K. Wang and B. Li. Efficient and guaranteed service coverage in partitionable mobile ad-hoc networks. In *Proceedings of IEEE INFOCOM 2002*, 2002.

[143] Karen Wang and Baochun Li. Group mobility and partition prediction in wireless ad-hoc networks. In *Proceedings of IEEE International Conference on Communications (ICC 2002)*, volume 2. New York City, New York, April 28 -May 2 2002.

[144] Ying-Hong Wang, Jenhui Chen, Chih-Feng Chao, and Tai-Hong Yueh. A dynamic caching mechanism for mobile ad hoc networks. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems - Workshops (ICPADS'05)*, pages 605–609, 2005.

[145] S.-C. Woo and S. Singh. Scalable routing protocol for ad hoc networks. *ACM Wireless Networks*, 7(5):513–529, September 2001.

[146] Bo Xu, Ouri Wolfson, Sam Chamberlain, and Yelena Yesha. Adaptive lazy replication in unreliable broadcast networks (extended abstract). In *7th Conference on Extending Database Technology (EDBT'2000)*, 2000.

[147] Y. Xue, B. Li, and K. Nahrstedt. A scalable location management scheme in mobile ad-hoc networks. In *Proceedings of the IEEE Conference on Local Computer Networks (LCN2001)*, Tampa, Florida, November 2001.

[148] Liangzhong Yin and Guohong Cao. Balancing the tradeoffs between data accessibility and query delay in ad hoc networks. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*, pages 289–298, 2004.

[149] Liangzhong Yin and Guohong Cao. Supporting cooperative caching in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(1):77–89, 2006.

[150] Hao Yu, Patrick Martin, and Hossam Hassanein. Cluster-based replication for large-scale mobile ad-hoc networks. In *International Conference onWireless Networks, Communications and Mobile Computing*, pages 552– 557, June 2005.

[151] Yinzhe Yu, Guor-Huar Lu, and Zhi-Li Zhang. Enhancing location service scalability with high-grade. In *IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2004)*, October 2004.

[152] Yinzhe Yu, Guor-Huar Lu, and Zhi-Li Zhang. Location service in ad-hoc networks: Modeling and analysis. In *Proceeding of NSF Workshop on Theoretical and Algorithm Aspect of Ad Hoc Wireless Networks*, June 2004.

[153] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.

[154] Jing Zheng, Jinshu Su, and Xicheng Lu. A clustering-based data replication algorithm in mobile ad hoc networks for improving data availability. In *Proceedings of 2nd InternationalSymposium on Parallel and Distributed Processing and Applications (ISPA 2004)*, pages 399–409, 2004.

[155] Lidong Zhou and Zygmunt J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, 1999.