

Data Management in Ad hoc Networks

Belgacem HLALI

Ecole Supérieure en Science Informatiques,
DEA Réseaux et Systèmes Distribués Sophia Antipolis, France.

Jean-Pierre HUBAUX and **Rachid GUERRAOUI**

Swiss Federal Institute of Technology, Lausanne
Communication Systems Department
Institute for computer Communications and Applications

march - august 2001

Acknowledgements

I would like to thank my advisor Prof. Jean-Pierre HUBAUX, for most helpful guidance. Special thanks to Mr. Rachid GUERRAOUI, who contributed to this work.

Table of Contents

1 Ad hoc Networks & The terminodes Project	5
1.1 Ad hoc Networks	5
1.2 Terminodes Project	5
1.2.1 Project description and benefits	6
1.2.2 Technological overview	7
2 The Storage Set : How does it operate?	9
2.1 Definition	9
2.2 The Storage Set	10
2.2.1 The Network model	10
2.2.2 The nodes	10
2.2.3 The algorithms	10
2.3 Transactions	11
3 Assumptions and Modeling	13
3.1 Network Model	13
3.1.1 Presentation	13
3.1.2 Crash and recovery model:	13
3.1.3 Replication Models	14
3.1.4 Complete Graph	14
3.2 Communications	15
3.3 FIFO (serializability)	15
3.4 Failures	15
3.5 The stored Data	16
3.6 Replication	16
3.7 Data Consistency	16
3.7.1 Consistency Methods	16
3.7.2 Updates methods	17
3.8 Replication Protocols and some existing systems	18
3.8.1 Maintaining consistency	18
3.8.2 Coda	18
3.8.3 Bayou	19
3.8.4 Ficus	19
3.8.5 Rumor	19
3.8.6 Roam	19

4 Algorithms in the storage set	20
4.1 Algorithms	20
4.1.1 Membership Service	20
4.1.2 Recovery Procedure	21
4.2 The Broadcast protocol	22
4.2.1 Broadcast Algorithms	23
4.2.2 Internal transactions	25
4.2.3 External transactions	26
4.2.4 proving the correctness of the algorithm	28
5 conclusion	29
A IEEE 802.11 : Summary	30
A.1 Architecture of IEEE 802.11 Networks	30
A.2 IEEE 802.11 Layers	31
B Bluetooth : Summary	33
B.1 Introduction	33
B.2 The Basic Bluetooth System Architecture	33
B.2.1 Overview of the Protocol Stack	34
B.2.2 Radio layer	34
B.2.3 Baseband	35
B.2.4 Radio layer	35
B.2.5 Broadcasting Data	35

Introduction

Data management is posing new challenging problems in wireless networks. In ad hoc Networks this need to data managing is due to not only increasing services, but also to the peer-to-peer relations between devices in such networks. Indeed, to be self organized, terminodes need to exchange and maintain some data between them.

Our work, part of terminodes project, consists of defining the essential primitives for the data managing in ad hoc networks and elaborating protocols to reach such a goal. In order to have a global view of the problem, we will be interested in different kinds of data such as private data (phone numbers,...), public data (news, announcements, ...) or shared data (fragments of files, databases,...).

In the first part of the project, we will study the possible strategies to store data in devices memories. Solutions such as replication scheme or fragmentation may be used. Of course, the choice of a solution depends exclusively of the kind of data to manage. Once stored, the data has to be maintained "alive" and managed in order to be reachable and coherent. So, in the second part will consist on finding the best way to manage the data and ensure the conditions we imposed. The availability may be ensured by making the data (public, or shared) independent of some specific terminodes in the network. For the data coherence, it's obtained by using adequate algorithms that guarantee (strict or tolerant) data consistency among all the copies.

Elaborating a distributed and integrated algorithm can represent the final part of the work. The algorithm will manage all the data queries (reading, writing, ..) and must be reliable, competitive and robust.

Chapter 1

Ad hoc Networks & The terminodes Project

1.1 Ad hoc Networks

In Latin, ad hoc literally means "for this," further meaning "for this purpose only," and thus usually temporary. The term has been applied to future office or home networks in which new devices can be quickly added, using, for example, the proposed Bluetooth technology in which devices communicate with the computer and perhaps other devices using wireless transmission. A mobile ad hoc network is *a self-organizing system of wireless nodes that requires no fixed communications infrastructure*. Each node is mobile, and equipped with a wireless communications device with which it exchanges packets with other nodes across one or more shared wireless channels. When direct communication between two nodes is not possible, packets are relayed through intermediate nodes in a multihop fashion to their destination.

1.2 Terminodes Project

TERMINODES is a 10-years (2000-2010) project which aims at studying and prototyping large-scale self-organized mobile ad hoc networks. The project distinguishes itself from other research projects in this area in several ways: first, it encompasses all layers and explores inter-layers interactions, from the fundamentals of the physical layer up to software architecture and applications; second, it is free from short term compatibility constraints such as the interoperation with IP networks (this interoperation will be studied in the project, however); last but not least, it tries to capture the business and societal potential generated by the paradigm shift previously described.

1.2.1 Project description and benefits

To a large extent, most of today's communication and information services are centralized. They require some basic underlying infrastructure without which the service cannot exist. In the case of information systems, base stations for mobile phones, backbone routers in the Internet and central databases are obvious examples. This basic infrastructure unfortunately often tends to become a source of increased cost, a bottleneck, a source of failures, and inevitably a deterrent to innovation. It also puts extremely valuable resources in the hands of the very few with the expertise or the economical power to control them.

As an alternative, we propose to leverage developments in mobile communication and mobile devices. This would give birth to a new kind of mobile information systems: a decentralized, self-organizing network based on (mobile) terminals that could simultaneously work as terminals for users, as well as network nodes connecting inter-user traffic. We call these multi-purpose terminals "terminodes", as the devices used to access the services are also part of the infrastructure. Or, in other words, the device used to communicate with a person is also a node relaying and routing messages in a vast network constituted by such devices. By extension, a portable computer becomes a server to any similar device within reach, thereby creating instant networks of information services.

Terminodes remove the separation between users and operators. By enabling completely decentralized information and communication services, terminode networks lay the foundation for seamless and ubiquitous communication. The potential benefits of a system based on terminodes are enormous, and so are the challenges associated with its development. It is thus only by pooling the resources and expertise of many individuals and institutions within a National Competence Center that will we give ourselves a chance to be successful.

The project Terminode extends the research currently existing in ad-hoc network (typically military application of infrastructure less network) and personal area network (as Bluetooth) by adding the scalability and performance issues and by extending the research at the information system and economic levels. Clearly, a Center of Competence on Mobile Information and Communication Systems will have a significant impact in many different areas:

- by focusing research efforts on leading-edge technology, it is bound to become a source of knowledge, innovation, and technological progress;
- the Center will make technology transfer a top priority. The interest

shown by many different companies (national, multinational and start-up; inside and outside Switzerland; technology and service oriented) demonstrates the need for this kind of research center. Links developed at an early stage between the Center and the industry will ensure that the research activities are truly in synch with the industry's needs and expectations;

- all the research activities carried out within the Center will lead to the generation of prototypes demonstrating both the feasibility of the ideas and their potential for commercialization. The emphasis on the practical aspects involved in the development of prototypes will help to train a new generation of graduate and undergraduate students. These will gain expertise in the development of complex, leading-edge information and communication systems. Over time, they will form a pool of valuable specialists that are likely to have a great influence in academia and the industry (e.g., in teaching, setting up of start-up companies, etc.);
- the Center will turn the involved institutions and Switzerland into one of the top places in the world for study, research, and career development in high technology.

1.2.2 Technological overview

The project aims at:

- leveraging developments in mobile communication and mobile devices; providing a new kind of network, relying on a decentralized, self-organizing system based on (mobile) terminals that could simultaneously work as terminals for users, as well as network nodes connecting inter-user traffic;
- popularize the concept of multi-purpose terminals "terminodes" as the devices used to access the services and part of the infrastructure. Or, in other words, the device used to communicate with a person is also a node relaying and routing messages in a vast network constituted by such devices. By extension, a portable computer becomes a server to any similar device within reach, thereby creating instant networks of information services.

The main research topics, which are planned, are as follows:

- Mathematical aspects: static and dynamic connectivity of terminode networks, achievable rates of transmission, quality of service
- Information-theoretic questions and physical layer: performance limits of low-power and/or wide-band communication systems, communication strategies and coding for multi-access and/or fading channels, multi-antennas, programmable testbed
- Networking aspects: neighbors selection and network connectivity, local and remote packet forwarding, location with/without GPS, information gathering
- Security: cryptographic algorithms, public key infrastructure, network robustness, individual privacy, management incentives
- Real-time applications and services, design of diversity-based communications protocols.
- Self-organized services in mobile environment: application services discovery, self-organized operating system, location aware services, distributed system on highly flexible infrastructure
- Architecture: tools and description techniques for performance-portable applications and middleware, integration of applications and services
- Embedded systems: low-power, small-size and large-population terminodes (keywords: smart labels, pico radio).

Chapter 2

The Storage Set : How does it operate?

2.1 Definition

We define the **storage set** as a group of nodes responsible for the management of a data between them. Depending on the strategy adopted, this data is stored in one or many nodes (replication). In a fully replicated scheme all the nodes of the storage set have a local copy of the data. The management of the data consists on ensuring :

- Availability: the group of nodes consisting the storage set must be able to respond to requests acceding to the data, and so make it available. The fully replicated scheme, increases the availability of the data since every node has the information and so can answer to different requests.
- Freshness: the data must be updated and every node should keep its local copy of the data as new as possible. This guarantees the coherence of the information delivered for the requests and avoid conflicts.
- Consistency: the consistency of the data is the fact that all copies should be coherent (the same) in all sites. It depends on the strategy chosen to ensure the consistency of the data (strict or tolerant) but a temporary inconsistency can be tolerated to reduce the complexity and simplify the protocols.

2.2 The Storage Set

2.2.1 The Network model

The general environment of this work is a wireless mobile network. Many models can be found : Master-Slave, Client-server, Peer-to-peer. The differences between these models and the choice of the appropriate model are mentioned in the next chapter. The choice of the appropriate must conserve the properties of ad hoc networks (auto-organization and any-to-any communications) and depends on the simplicity that offers to implement protocols for the data management.

2.2.2 The nodes

The nodes of the network are the different Mobile stations. There two ways to classify these devices :

- belong or not to the storage set: the nodes that belong to the storage set (*members*) are responsible for the management of the stored data (independently if they have a copy of the data or not). The rest of the network nodes don't belong to the storage set.
- failed or not: For the storage set members we distinguish between the cases if the device is operational or failed. Many types of failures can be imagined in the model we studied and will be discussed in the next chapter.

2.2.3 The algorithms

In this paragraph, we will present some algorithms used in the storage set in order to realize the data management and to ensure its operational mode.

The Broadcast Algorithm

Such algorithm is developed to ensure the organization of the storage set (communications, transactions, ...). This algorithm must exploit the property of wireless networks : the native broadcast (a message is received by all neighbors). There are many types of Broadcast algorithms and the general classification of them is *Reliable Vs Unreliable* Broadcast. Reliable broadcast algorithms guarantee some properties (Validity, Agreement and Integrity) which will be explained in the next chapter. Reliable algorithms may also guarantee the Total order of messages, the Fifo order of messages issued from one node, uniformity

Membership service & Failure detection

The role of a membership service is to maintain the list of the current active members of the storage set. This algorithm is needed since we consider possible failures of the devices. To establish such a list, it must be possible to detect failures.

The recovery Procedure

Since we consider that devices can fail for while, after what they can reappear in the network and integrate the storage set. In this case a specific procedure is invoked to correctly integrate the storage set. This procedure must ensure the update of local copy (if the failed member has a local copy), and informs the other members for this recovery.

2.3 Transactions

Typically, transactions are *read* and *write* commands. These commands are launched by different nodes of the network (may belong or not to the storage set). A node sends a read command, when it needs an information from the stored data. A node emits a write request in order to update the stored data.

- **Write operations:** These operations, since they allow modification of the data, can generate some inconsistencies if not performed adequately. Of course, some systems can tolerate a temporarily data inconsistency to increase the availability. These *optimistic replication systems* allow any machine to perform an update locally, rather than requiring the machine acquire locks or votes from other replicas. Update propagation can be attempted immediately when the update occurs, or at a later time. *Immediate propagation* notifies other replicas of the new state of the data as quickly as possible. Alternatively, updates can be propagated at a later, more convenient time, typically batched. This option of *periodic reconciliation* does not spread updates as quickly, but allows propagation to occur when it is cheap or convenient. External write requests, those sent by nodes which don't have a direct access to the data, must be performed by an intermediate storage set member. This member (a specific algorithm is introduced for this purpose of choosing a member) receive the request (after a connection establishment with the initiator) and execute it as an internal write command.
- **Read operations:** The read operations consist on requests to access to a information in the storage set. There's a difference if the node

sending the request possesses a copy of the data or not. If it is the case, the read request is proceeded locally (access to the local copy). Otherwise, the sender doesn't have a local copy, it must access to a distant copy. The difficulty of executing such commands is how to designate who will replay to that request?. An appropriate algorithm must perform and indicate which member should do that (an election algorithm, randomly, primate copy, ...). Of course this member must be able to communicate with the request initiator.

Chapter 3

Assumptions and Modeling

Introduction

In this chapter, we will discuss the assumptions adopted for modeling the storage set. These assumptions not only the network but also the data and the general organisation of the storage set. We will begin by modeling the storage set. After that, general aspects of the storage set are discussed. Finally, the assumptions concerning the data are announced.

3.1 Network Model

3.1.1 Presentation

We will use graphs to model the stage set. The whole Network is represented by a graph where nodes are the vertices and links are the edges between them. In a wireless environment, links are radio channels used for communications. The storage set is a part of this graph.

3.1.2 Crash and recovery model

In this model processes can disappear (crash) and may reappear (recover) after a while. It seems to be the most appropriate model for the storage set because the devices can crash. The crash is not only due technical, interference or weather conditions but also by a simply turn off performed by users. Devices may recover after a while, and integrate the storage set. The behavior of the storage set composition depends on general conditions of the network and members profiles.

3.1.3 Replication Models

There are three basic models : master-slave, client-server and peer-to-peer.

- **Master-slave model** : The master-slave model labels one replica the "master"; all other replicas are slaves. Slaves should always be identical to the master. The model is very simple, and has been used in many replication packages such as LAPLINK and RDIST. However, it has limited functionality : the slave is essentially read-only.
- **Client-server Model** : The client-server model is similar to the master-slave in that it designates one server which serves multiple clients. However, the functionality of the clients is greatly improved, and multiple inter-communicating servers are permitted. Clients can generate all types of data modifications and updates. The client-server model has been successively implemented in systems such as CODA and LITTLE WORK. The major drawback for this model is the that clients cannot intercommunicate or synchronize with each other.
- **Peer-to-peer Model** : The peer-to-peer model takes a very different from the both master-slave and the client-server models. The peer model is not a class based: all replicas are equals, or peers. Any replica can synchronize with any other replica, and any modification or update can be applied at any accessible replica. The peer model has been implemented in systems such as LOCUS, BAYOU, FICUS and RUMOR. This model provides a very rich and robust communication. However, it has typically suffered from scaling problems.

In the case of the Storage set model, the peer-to-peer is the suitable one with regard to the general properties of Ad hoc networks (any-to-any communication).

3.1.4 Complete Graph

In the graph theory a complete graph is one for which every two nodes of the graph have a link between them. It's the same assumption for the storage set where every active member is reachable by all the others. This means that every message sent by a storage set member arrives to all the other nodes without a relay or forward function. It's a strong assumption, which can be accepted only when members of the storage set are close to each others and also can establish any-to-any communications (peer-to-peer).

3.2 Communications

The assumption for inter-terminodes communications is the synchronous one. In opposite to asynchronous communication, where no knowledge of time limits on the time it takes for a message, synchronous communication suppose that any message sent by a device is always received within a given delay. Synchronous assumption is used because members are supposed not so far from each others and in order to establish simple protocols for data management.

3.3 FIFO (serializability)

First-in-first-out is the technique adopted in the model of the storage set. In the message queues of each node, commands are executed in the order of which they arrived, of course after a possible reordering to respect causality requirements and to ensure data consistency. The problem of reordering happens, essentially, when two processes send a lock request of the same data, and these requests arrive in a different order in the others sites. So a reordering procedure must be activated to reorder the messages and avoid conflicts.

3.4 Failures

failures in mobile environments are very frequent. Three kind of failures can be imagined :

- A member can fail by simply crashing and sends no messages before its recovery.
- A member can fail by sending erroneous messages.
- The network can partition to more than one component.

In the storage set model we will consider, for a primary study, only failures when a device is voluntarily turned off. For such failure it's possible that message is sent by a node to inform the others about its disconnection. For a deeper study, the first kind of failures where a device simply crashes and send no message before its recovery, may also be considered in order to make more realistic and robust algorithms.

All the assumptions above are relative to storage set nodes organization. Now, we will see the considerations for the Data to manage :

3.5 The stored Data

Concerning the data to manage between the nodes of the storage set, we will not be interested in a special kind of data such as private data (phone numbers,...), public data (news, announcements, ...) or shared data (fragments of files, databases,...). We will, only assume that every node in the network is responsible for a particular part of the stored data.

3.6 Replication

Replication consists on placing different copies of the data locally in some nodes. It is especially important in mobile environments, since disconnected or poorly connected machines must rely primarily on local resources and so need to have important data replicated locally. It is used to improve reliability, availability and data access performance. The replication scheme determines how many replicas are created and to which devices these replicas are allocated. The scheme is *static* when it remains fixed until manual reallocation is performed. And it is *Dynamic* when it changes as the configuration of the network and the read-write pattern of data changes in time. The last scheme is more suitable for mobile environments.

In the storage set model we use a fully replicated scheme, where a copy is created at each node of the group. This scheme is very interesting in the case of dominating read pattern, since all read operations are proceeded locally. The main drawback of such scheme is the write commands. When performed, these operations must be broadcast to all other copies to maintain the data consistency.

3.7 Data Consistency

Typically, in replication context, data is consistent when all copies are the same. Since write operations are the only possibility to violate this condition, they must be performed carefully. Messages ordering in the queues of nodes is used to ensure that operations are executed in the same order in all node and avoid conflicts.

3.7.1 Consistency Methods

Consistency can be maintained *optimistically*, *conservatively* or with a mixture of both approaches.

- **Conservative Method** : Conservative schemes use some form of locking, voting, or primary-site techniques to prevent conflicting updates. These schemes guard against all concurrent updates, but in doing so reduce data availability. If an update cannot be written because a lock cannot be obtained or a majority of other sites cannot be queried, the data object is effectively unavailable. Since disconnections and network partition are normal occurrences in mobile, conservative strategies are not a viable solution.
- **Optimistic Method** : Optimistic method assumes that concurrent updates, or conflicts, are rare. It allows updates to be performed independently at any replica. When conflicts do occur, a *conflict resolution* procedure must be executed to resolve the conflicts and merge the concurrent updates into a single data object. This approach is difficult to implement and can potentially suffer due to high volume of conflicts. However it is attractive since it provides high availability of the system, which is precisely what we require.

3.7.2 Updates methods

Modification made in on replica can be propagated immediately, *immediate propagation*, or periodically, *periodic reconciliation*.

- **Immediate Propagation** Update propagation-based replication attempts to propagate updates made at one replica to the other replicas immediately. In a frequently disconnected environment, some or all of these update propagations are doomed to failure. Indeed, resources are scarce and expensive, perhaps when immediate propagation was not very important.
- **periodic reconciliation** Alternatively, in a reconciliation-based replication, no attempt is made to propagate updates automatically. Instead, periodically all changes made to the replicated data are batched together and sent to the other copies. These batched changes can be sent during periods of high, cheap connectivity.

In our storage set model, update messages are queued in the nodes, and there's no synchronization to execute these commands in the same time. So a temporary inconsistency may occur in the Data. This let nodes operate independently from each others in a distributed manner.

3.8 Replication Protocols and some existing systems

3.8.1 Maintaining consistency

The problem of maintaining the consistency of the data, in a replication scheme, is known as multiple copy update problem. Solutions to the multiple copy update problem can voting or non-voting.

- Non-voting solutions: (are non-democratic)
 - The primary site approach: based on the concept of centralization. This is very simple approach because detecting and resolving transaction conflict centrally can be made easy. The drawback is the low reliability (if the controller fails, all the total system crashes)
 - The token passing approach: a token is in charge of making all sequencing and synchronization decision.
- Voting solutions: (result of negotiations, decentralized)
 - Majority voting: when updating a process must obtain the consent of at least a majority to proceed the transaction, else it's not allowed until some later time. Voting algorithms Thomas (79), and Bernstein (85). These 2 algorithms cannot be used for performance reasons (complexity and overhead).
 - Weighted voting: (proposed by Gifford 1979). This algorithm can be characterized informally in the following way : Firstly, 1. each copy is assigned some number of votes. Then, to perform a read operation a transaction must collect a read quorum of r votes, and w for a write operation. $r + w$ must be superior than the total of votes allocated to that process.

3.8.2 Coda

the Coda file system is an optimistically replicated file system constructed on a client-server model. The rigidity of the model dramatically simplifies the consistency algorithms at the cost of limiting the system's utility for mobility.

3.8.3 Bayou

It is based on the peer-to-peer model and provides support for application-dependent resolution of conflicts. It does not provide any form of *selective replication*. The data must be entirely replicated at all storage sites.

3.8.4 Ficus

Like Bayou, it's based on the peer-to-peer model and provides selective replication. It maintains consistency with a periodic reconciliation process, and uses a best-effort, real-time propagation of updates. However, it is unsuitable system for mobile use, and suffers from scaling problems.

3.8.5 Rumor

Rumor is the direct predecessor of Roam and shares the same characteristics and problems as Ficus (peer model, and scaling problem) since it was designed to be a direct adaptation of Ficus, only at the user level instead of integrated in the kernel.

3.8.6 Roam

Roam is built using the *Ward Model*, a replication architecture that has been designed especially with mobility in mind. The ward Model is a hybrid model of peer and client-server that allows everyone to be effectively a peer while maintaining good scalability in the number of replicas. Roam provides :

- An optimistic replication system capable of supporting both stationary and mobile users
- The ability for any-to-any communication and synchronization between any two replicas in the entire system
- An architecture that scales well

Chapter 4

Algorithms in the storage set

In this chapter, we will present all algorithms needed for the data management in the storage set. These algorithms are given with the different possibilities. After that we will present our choices.

4.1 Algorithms

In this section we will enumerate the different possible solutions to realize each component of the operating mode of the storage set. We will begin by the membership service and the recovery procedure. After that we will present the Broadcast algorithm useful to execute different transactions.

4.1.1 Membership Service

Such algorithm should maintain the list of active current members of the storage set. The view given by the algorithm must be the same to all nodes. In what follow, we will comment some possible solutions and present our choice.

- Each node, verify the current active members, by sending queries : "*Are you alive*" messages. members who don't answer to that query are automatically removed from the list. The main drawback of this solution is the overhead of the network. About n messages are needed for each node to establish the list of the current active members (1 sent message :the query and $n - 1$ replays). Another problem is How to ensure the same view of the group by having an identical list in all nodes, since each node acts independently.
- Instead of sending queries and waiting for responses, periodic "*I'm alive*" messages can be sent by each node. On receiving such message

the sender is added (or maintained) on the list of active members. To detect failures each node maintains timers for all other members and if no "I'm alive" message was received after a timeout then, the member is considered as failed. The main problem of such approach how to tune the period? to make it as optimal as possible. And after how many period a member is notified as failed? to avoid erroneous detections.

- A centralized approach may be used. In this approach, one copy is made responsible for that service and delivers the list to all members. The responsible node may send queries ("*are you alive*" messages) to the other members to detect if they are still active or not. Or simply, the members send informing messages ("*I'm alive*" messages) to the primary node which establishes the list. This centralized approach doesn't really go with the general aspect of ad hoc networks in which nodes are equivalent and relations are peer-to-peer. This makes the primary node as a bottleneck and a failure of such node is fatal for the operating of the storage set.

Comparing the above approaches, the second one seems to be the most appropriate. But, in our primary study we will consider only voluntary turn off failures. In this case, before a disconnection, a member informs the other members by sending a specific message.

4.1.2 Recovery Procedure

When a member recovers after a failure, it should not only integrate the the active members list other and inform nodes of its recovery but also update the local copy it has. To integrate the list of active members, the recovering node can simply send a notification message (it can be an "I'm alive" message). So every node receiving such message will add the new member to the list. To make an update of the data the new node must send a request for that (can be piggybacked in the announce message). In that message is mentioned the sequence number of the last version of data it has. If there's a newer version, the other members should manage themselves to designate someone who will send a copy of the data. The choosed member establishes a point-to-point connection and proceeds to the transfer of the data, the messages in the queue and the list of members. Now, we will discuss how to designate that member?. For that many approaches are possible:

- The nearest: each node calculates the distance from the new member. Messages containing the calculated distance are exchanged and the nearest knows itself and carry out the transfer. The problem for

this approach is the processing which is needed to calculate distances (energy and time consumption), the overhead of messages (on average $\frac{n}{2}$ messages are exchanged) and also that it supposes the nodes have localisation information which is not true in all cases.

- Token passing: It's a centralized approach, in which a token is circulating between the members of the storage set. The member having the token, carry out the transfer. The drawback of such approach is the centralization in which the node having the token becomes the bottleneck of the network and the difficulty to manage the token (what to do when the token is loosed).
- Election: the member having the newest version is elected. For this approach about n messages are exchanged to know the node having the recent copy of the data. Otherwise, if a table containing the last version number of each node is maintained, then the member having the newest copy knows itself and carry out the transfer.
- Random approach: this method operates as the following way. On receiving the announcement for the recovery, every member of the storage set randomly chooses a period T after which, it will carry out the data transfer. The node having the smallest period will be the first to send a broadcast message, so the others are aware of this answer and simply will abort their possible replays. This a simple approach, but the problem persists when two nodes choose the same period T . This happens with a very small probability.

The last seems to be the most appropriate, when the table containing the last version numbers of all nodes is maintained. This not only due to the simplicity of its implementation but also the fact the new is the version the small is the size of data to carry out (when a version A is newer than B , this means that node B processes more messages than A . And so A contains more message waiting in its queue file)

4.2 The Broadcast protocol

In this paragraph we will discuss how to execute transactions in the storage set. Using a broadcast protocol, these transactions are conveyed and transmitted between nodes. before presenting our algorithm, we will give a small overview of the broadcast algorithms. Fault-tolerant broadcasts are communication primitives that facilitate the development of fault-tolerant applications. The weakest among these is *Reliable Broadcast*. Roughly speaking,

this allows processes to broadcast messages such that all processes agree on the set of messages they deliver, despite failures. Stronger variants of Reliable Broadcast impose additional requirements on the order in which messages are delivered. Before presenting the different broadcast algorithms three properties are introduced to make the difference between them. There three main properties:

- *Validity*: If a correct process broadcasts a message m , then all correct processes eventually deliver m .
- *Agreement*: If a correct process delivers a message m , then all correct processes eventually deliver m .
- *Integrity*: For any message m , every correct process delivers m at most once, and only if m was previously broadcast by the sender of m .

4.2.1 Broadcast Algorithms

Reliable broadcast

The reliable Broadcast is the weakest type of fault-tolerant, no restriction on the order. It requires that all correct processes deliver the same set of messages (Agreement), and that this set include all messages broadcast by correct processes (validity) but no spurious messages (integrity). Formally, the reliable broadcast is defined in terms of two primitives: **broadcast** and **deliver**. Validity together with Agreement ensures that a message broadcast by a correct process is delivered by all correct processes. Every message is unique and includes the identity of the sender and a sequence number. If the sender of m is faulty, the specification of reliable broadcast allows 2 outcomes: either m is delivered by all correct or by none.

FIFO Broadcast

It is a reliable broadcast that satisfies the following requirement on message delivery.

FIFO order: if a process broadcasts a message m before it broadcast m' , then no correct process delivers m' unless it has previously delivered m .

Fifo order is adequate when the context of a message m consists only of the messages that the sender of m broadcast before m . A message m , however, may also depend on messages that the sender of m delivered before broadcasting m . In this case, the message delivery order guaranteed by FIFO Broadcast is not sufficient.

Causal Broadcast

The Causal Broadcast is a strengthening of FIFO broadcast, messages delivered according to the causal precedence relation.

Causal order: if the broadcast of m causally precedes the broadcast of m' , then no correct process delivers m' unless it has previously delivered m .

Local order: if a process broadcasts m and a process delivers m before broadcasting m' , then no correct process delivers m' unless it has previously delivered m .

Atomic Broadcast

If the broadcast of two messages are not related by causal precedence, Causal Broadcast does not impose any requirement on the order they can be delivered. In particular, two correct processes may deliver them in different orders. To prevent such problems Atomic Broadcast requires that all messages are delivered in the same order by all correct processes. So all processes have the same "view" of the system (consistency is ensured without additional communication).

Total order: if correct processes p and q deliver m and m' , then p delivers m before m' if and only if q delivers m before m' .

FIFO Atomic Broadcast

FIFO Atomic Broadcast is a Reliable Broadcast that satisfies both FIFO Order and Total Order. FIFO Atomic Broadcast is stronger than both Atomic Broadcast and FIFO Broadcast.

Causal Atomic Broadcast

It's a Reliable Broadcast that satisfies both Causal Order and Total Order. Causal Atomic Broadcast is stronger than both FIFO Atomic Broadcast and Causal Broadcast.

Timed broadcasts

D-Timeliness, is the following property: if a message is delivered, it is delivered within a bounded time after it was broadcast. Two ways to define this property, depending on the use of local or real time. D is the latency.

Uniform broadcast

In the above broadcasts there's no restriction on messages delivered by faulty processes (agreement allows a faulty process to deliver a message that is never delivered by the correct processes).

- **Uniform Agreement:** if a process (correct or faulty) delivers a message m , then all correct processes eventually deliver m . Similarly, Integrity allows a faulty process to deliver a message more than once.
- **Uniform Integrity:** for any m , every process delivers m at most once, and only if m was previously broadcast by sender(m). We can also strengthen to uniform D-Timeliness. Likewise, we can strengthen each of Order properties, by requiring that even faulty processes do not violate them. Theorem: uniform causal order is equivalent to uniform FIFO Order and Uniform Local Order.

According to the assumptions adopted to model the storage set, where each part belongs to a particular node in the network. And this node is responsible for possible updates and modifications of it. The FIFO Reliable Broadcast is the algorithm chosen to execute broadcast primitives (*read* and *write* transactions). So the developed algorithm should guarantee the FIFO Order in messages sent by the same sender. There's a difference between primitives sent by nodes that belong to the storage set (internal transactions) and those emitted by nodes that do not belong to the storage set (and do not have a copy of the data file). We will begin by studying the broadcast of internal transactions.

4.2.2 Internal transactions

Since, we are using a fully replicated scheme, where all storage set members have a local copy of the data, only write commands are broadcast in order to keep all copies identical. We choose a Uniform FIFO Reliable Broadcast. This protocol intervenes to execute transactions (internal and external). So when a node modifies its local copy, it must broadcast this modification to other copies of all active members. A kind of carrier sensing can be used to try to avoid possible collision. We say, try to avoid, since a perfect collision avoidance in wireless is not easy to establish without RTS/CTS and due to the "hidden node" problem. Two possible improvements of this protocol can be added:

- A simple request to send is emitted by the initiator node before the broadcast. In this RTS-like, the node informs the others that it will

broadcast a message after a time Δ (randomly choosed). So the other nodes will be aware of the future broadcast and will schedule their traffic in order to avoid interference.

- Possible multi-transmission of the same message may be used in order to increase the reliability of the protocol. the number of retransmissions is tuned regarding the load of the network, interference, the number of active nodes and the weather conditions.

Messages, before their delivery, are queued in nodes. To order messages we use a kind of version number, which is incremented at every update (write operation). So messages are ordered with a growing version number. If two messages have the same version number, they are ordered as they arrived to the node. In a first view, we may say this version number can guarantee also FIFO conditions. This is true, if there's no lost of messages and that all messages arrive. But if messages can be lost, this can violate FIFO conditions. Let's take an example. If a node A, broadcasts 3 messages m_1, m_2 and m_3 and the second message was lost. In this case, only messages m_1 and m_3 are delivered (m_2 was lost). Using only version number, the lost can't be detected. So, a sequence number is introduced and this guarantees the FIFO conditions and also detect possible losses. Every node maintains a table containing the sequence number of the expected message from each. And a message is delivered only if its sequence number corresponds to that expected. Otherwise, it's queued.

```

Ibdcst: in one time send to all
Broadcast(member_identity, set_identity, update, m)
my_version # = ++
On receive:
if { queue  $\neq \phi$  }
then order with respect to seq# and version#
On delivery:
if seq[sender] = seq# then
seq[sender] = ++
deliver
my_version# = ++
remove
else Next

```

4.2.3 External transactions

The problem with these transactions that initiator nodes, which don't have a local copy of the data, when they broadcast such requests, they may reach

only some members of the storage set. So, in the case of a write request, the query must be rebroadcast (by a member of the storage set) to reach all members and to guarantee the data consistency. let's begin with read requests.

External read transactions

When a node which doesn't belong to the storage set wants to accede to an information from the replicated data, it emits a read request. Since, all nodes of the storage set has a copy of the Data, any one which receives the request may satisfy it. To define which node will answer to the read request, we will exploit the fact that all storage set members are permanently listening to each others. So, the first node answering to the request will make all the others aware about that answer. And so all the other nodes will cancel their possible replay. To avoid, that nodes broadcast replay message in the same time a random function is introduced. Such function chooses randomly a variable Δ . And after the time Δ a node can answer to the read request if none has already replayed.

External write transactions

The requirement for such commands are more strict since they act on modifying the data. The treatment is almost similar to the read request. But the type of the replying message is different, of course. The first node responding to that request, doesn't send a message to the sender, but broadcast an update in order to reach all members. This update, is therefore treated as an internal one.

Ebcast : External Read and Write Requests
Sender: *Broadcast Read_rq (sender_identity, set_identity, @data, last_version #)* or
Write_rq (sender_identity, set_identity, @data, last_version #)
A Node on the storage set:
On receive
if my_version \geq last_version (to guarantee FIFO order)
then
 $\Delta = \text{Random}(T)$ randomly choose Δ between 0 and T
Wait Δ
if no receive *Answer(Ext_rq (sender_identity, set_identity, @data, last_version #))*
then Bdcst *Answer(Ext_rq (sender_identity, set_identity, @data, last_version #))*
else (a message is sent as a response to that query)
IF write_rq **then** queue the message
Else remove *Answer(Read_rq (sender_identity, set_identity, @data, last_version #))*
remove *Ext_rq (sender_identity, set_identity, @data, last_version #)*

4.2.4 proving the correctness of the algorithm

As we said, the algorithm has to satisfy, reliability, uniformity and FIFO properties. To be a reliable broadcast, the algorithm must ensure :

- Validity (if a correct process broadcasts a message m , then it eventually delivers m): If a node broadcasts a message, it will receive it and so eventually delivers it. This ensures the validity property.
- Integrity (For any message m , every correct process delivers m at most once, and only if some process broadcast m) : An internal message is unique and broadcast in one time. Then its received, by all correct nodes, only once and so delivered once. For an external request, the integrity is ensured by the fact that nodes (which receive the original request) choose randomly the time when to respond. The first node responding to that request will make aware all the others and so they cancel their possible responses. Again, the unicity of the response ensures that the message is broadcast once and so delivered at most once.
- Agreement (If a correct process delivers a message m , then all correct processes eventually deliver m) : Every message is delivered only if its broadcast by a member of the storage set. For an internal message, it will reach all correct members and so delivered by all correct. If its an external message, its delivered only if rebroadcast by a member of the storage (the first responding), and so reach all correct members and so delivered by all correct.
- FIFO: Since every message is assigned a sequence number and a table is maintained in the nodes for those numbers, a message is delivered if and only if its sequence number is exact. Otherwise its queued. This guarantees the FIFO order on messages delivery.
- Uniform agreement (If a process delivers a message, every correct process delivers the message): If a process receives the message then all other active members will receive it and so deliver the message. This remains true only if we don't consider network partitioning failures.
- Uniform integrity (Every process delivers a message at most once, and only if the message was previously broadcast): Every message is unique and broadcast once and so received once.

Chapter 5

conclusion

In this report, we have studied the problem of data management in ad hoc networks. The problem was to maintain some data between a group of nodes called *storage set* with guaranteeing some properties (data availability, data consistency, ...).

We begun by modeling the storage set and defining the different possible primitives on the stored data (*read* and *write* primitives). In the second part we enumerated the different algorithms and procedures necessary to realize the management of the data (membershiop service, recovery procedure). After, we discussed the different possible solutions for each procedure. Finally, we made our choices and the result is a Reliable Broadcast algorithm. This algorithm is used to execute the different transactions acting on the stored data to maintain. According to the assumptions we made, one node is responsible for the modifications of a particular part of the data, the algorithme guarantees the FIFO Order in transaction issued from one sender.

Although the algorithm is made simple by the assumptions we adopted, but it's important not only in the special case of these assumptions but also as a beginning study for the problem of data management in ad hoc networks. This work can be extended to be operational on existent WLANs such as Bluetooth or IEEE 802.11, and also by making general and more realistic assumptions on the network and the kind of data to manage.

Bibliography

- S. AHARYA and A. B. ZDONIK, "An efficient scheme for Dynamic Data Replication", Department of Computer Science, Brown University, September 1993.
- J. BRAY and C. F. STURMAN, "BLUETOOTH : Connect Without Cables", Prentice Hall PTR, Upper Saddle River, New Jersey 2000.
- Kwang-Cheng CHEN, "Medium Access Control of Wireless LANs for Mobile Computing", IEEE Network September/October 1994.
- J. GRAY, P. HELLAND, P. O'NEIL and D. SHASHA, "The Dangers of Replication and a solution"
- V. HADZILACOS and S. TOUEG, "A Modular Approach to Fault-Tolerant Broadcast and Related Problems", Department of Computer Science, Cornell University, NY USA.
- Y. Huang, O. Wolfson, "A Competitive Dynamic Data Replication Algorithm", Electrical Engineering and Computer Science Department, University of Illinois at Chicago.
- I. Keidar and D. Dolev, "Efficient Message Ordering in Dynamic Networks", Computer Science Institute, The Hebrew University of Jerusalem.
- B. KEMME and G. ALONSO, "A Suite of Database Replication Protocols based on Group Communication Primitives", Information and Communication Systems Research Group, Institute of Information systems, Swiss federal Institute of Technology (ETH).
- A. LUBINSKI and A. HEUER, "Configured Replication for Mobile Applications", Computer Science Dept., University of Rostock, Germany, 2000.
- N. A. LYNCH, "Distributed Algorithms", Morgan Kaufmann Publishers, Inc. San Francisco, California.
- D. H. RATNER. "Roam : A Scalable Replication System for Mobile and Distributed Computing", University of California, Los Angeles, January 1998.
- D. RATNER, P. REIHER, J. POPEK and G. H. KUENNING, "Replication requirements in Mobile Environments", Department of Computer Science, University of California, Los Angeles.

- P. REIHER, J. POPEK, M. GUNTER, J. SALOMONNE and D. RATNER, "*Peer-to-peer Reconciliation Based Replication for Mobile Computers*", ULCA.
- R. ROZOVSKY and P. R. KUMAR, "*SEEDEX : A MAC protocol for ad hoc networks*", Dept. of Electrical and Computer Engineering, University of Illinois, USA.
- I. Stonoi, D. Agrawal and A. El Abbadi, "*Using Broadcast Primitives in Replicated Databases*", Department of Computer Science, University of California, October 1997.
- B. H. WALKE, "*Mobile Radio Networks : Networking and protocols*", John Wiley & Sons, England 1999.