الجمهورية الجزائرية الديمقراطية الشعبية
People's and Democratic Republic of Algeria
وزارة التعليم العالي و البحث العلمي
Ministry of Higher Education and Scientific Research
جامعة عمار ثليجي بالأغواط
University of Ammar Telidji Laghouat

كلية العلوم

Faculty of Science

Thesis presented by:

Tahar ALLAOUI

For the Doctor of Science degree

Specialty: Computer Science

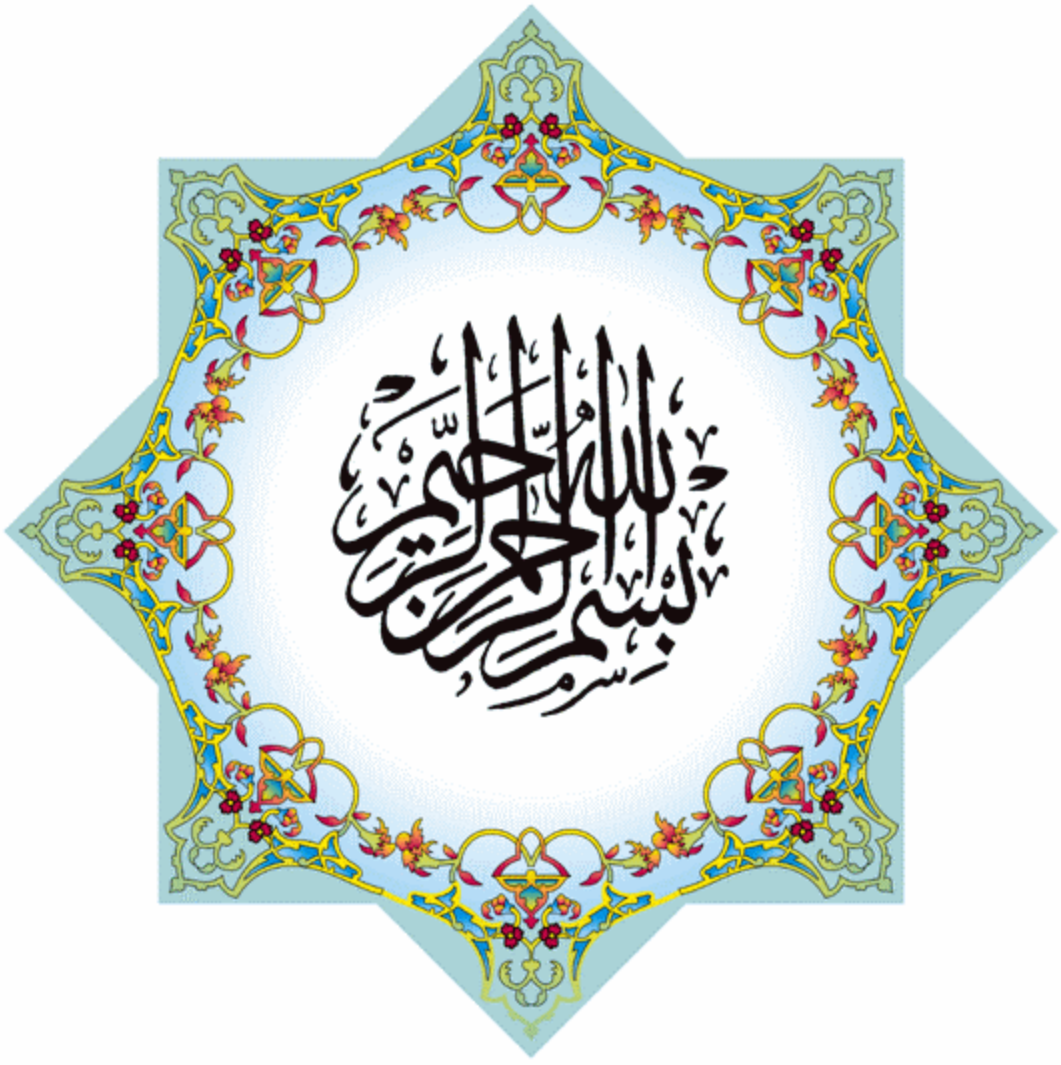
Theme:

Resources Sharing and Fault Tolerance in Distributed Systems

Publicly defended in September the 21st 2020 before the jury composed of

Mr. Nasreddine LAGRAA	Professor	University of Laghouat	President
Mr. Mohamed BENMOHAMED	Professor	University of Constantine 2	Examiner
Mr. Abdou elkarim TAHARI	MCA	University centre of Aflou	Examiner
Ms. Zohra ABDELHAFIDI	MCA	University of Laghouat	Examiner
Mr. Mohamed Bachir YAGOUBI	Professor	University of Laghouat	Advisor

2019-2020


بسم الله الرحمن الرحيم

سُورَةُ الأَعْرَافِ ٧

Dedications

I dedicate this work to

My dear parents,

My dear wife,

My dearest son Ahmed Wael,

The whole family, especially little Imad,

All my friends,

Acknowledgements

In the beginning Alhamdulilah,

I would like to address my most sincere thanks to Professor Mohamed Bachir YAGOUBI, who as a thesis director has always been attentive and very available throughout the realization of this work, as well as for the inspiration, help and time that he has kindly consecrated to me throughout my post-graduation process.

I would like to thank the members of the jury.

I also would like to thank all the people who have helped me in any way to accomplish this work.

ملخص

في هذه المذكرة نهتم بدراسة اشكالية استعمال الموارد المتعددة و التي تعتبر امتدادا لمشكلة استعمال الموارد بصفة حصرية في الانظمة السلكية و اللاسلكية.

في هذا الإطار ينقسم بحثنا إلى ثلاثة أقسام رئيسة، قسم نظري و قسمان تطبيقيان.

في القسم الأول نستعرض المفاهيم الأساسية المتعلقة بأنظمة الإعلام الآلي الموزعة و نحاول فهم إشكالية الإستخدام الحصري للموارد المفردة و المتعددة، كما ندرس الحلول المقترحة، كما نقدم مختلف الطرق المتبعة لمواجهة الأعطال في الأنظمة الموزعة و الشبكات اللاسلكية.

في القسم الثاني من البحث نقترح حلا جديدا لمشكلة الإستخدام المتعدد للموارد في الأنظمة السلكية، مستخدمين طريقة مبتكرة تسمح بتقليل الأعباء المتعلقة بتبادل الرسائل، كما نعزز هذا الحل بطريقة تخول له العمل في وجود الأعطال التي تؤثر على السير الحسن للنظام.

أما في القسم الثالث، فسوف نقترح طريقة جديدة أيضا لحل نفس المشكلة ولكن هذه المرة في الشبكات اللاسلكية، كما نعزز هذا الحل بطريقتنا المبتكرة التي تسمح له بالعمل في وجود أعطال على مستوى مواقع للنظام.

الكلمات المفتاحية: الأنظمة الموزعة، الشبكات السلكية و اللاسلكية ، استعمال موارد النظام، مقاومة أعطال النظام

Abstract

This thesis deals with the problem of K-mutual exclusion that can be seen as a generalization of the problem of simple mutual exclusion in distributed systems and mobile ad hoc networks.

Our work is divided into three parts, one theoretical part and two practical parts.

In the first part, we first introduce the basic concepts of distributed systems and explain the problem of mutual exclusion and K-mutual exclusion, while showing how these problems have been solved. We also present the notion of dependability, and the fault tolerance methods used to ensure it.

In the second part, we propose a new solution to solve the problem of K-mutual exclusion; we explain its operating principle and the new logical structure used. Our solution is based on the use of tokens, and allows ensuring K-mutual exclusion with a reduced number of exchanged messages. This new solution will then be improved by a fault tolerance mechanism that allows the system to continue to operate even in the presence of faults.

In the third part of this thesis, we present a new solution to the problems of K-mutual exclusion in mobile ad hoc networks, which exploits routing protocols to ensure resource sharing. This algorithm will also be improved by a powerful fault tolerance protocol.

Future perspectives and improvements are proposed at the end to refine these new solutions.

Key words: Distributed systems, Resources sharing, Mutual exclusion, K-mutual exclusion, Fault tolerance, Mobile Ad hoc networks.

Résumé

Ce mémoire de thèse traite le problème de la K-exclusion mutuelle qui peut être vu comme une généralisation du problème de l'exclusion mutuelle simple dans les systèmes répartis et les réseaux mobiles Ad hoc.

Notre travail est divisé en trois parties, une partie théorique et deux parties pratiques.

Dans la première partie, nous introduisons d'abord les concepts de base des systèmes répartis et nous expliquons le problème de l'exclusion mutuelle et de la K-exclusion mutuelle, tout en montrant comment ces problèmes ont été solutionnés. Nous présentons également la notion de la sûreté de fonctionnement, et les méthodes de tolérances de fautes utilisées pour assurer cette sûreté.

Dans la deuxième partie, nous proposons une nouvelle solution pour résoudre le problème de la K-exclusion mutuelle, nous expliquons son principe de fonctionnement et la nouvelle structure logique utilisée. Notre solution est basée sur l'utilisation de jetons, et permet d'assurer la K-exclusion mutuelle avec un nombre réduit de messages échangés. Cette nouvelle solution sera ensuite améliorée par un mécanisme de tolérance aux fautes qui permet au système de continuer à fonctionner même en présence des fautes.

Dans la troisième partie de ce mémoire, nous présentons une nouvelle solution aux problèmes de la K-exclusion mutuelle dans les réseaux mobiles Ad hoc, cette solution exploite les protocoles de routage pour assurer le partage de ressources. Cet algorithme sera également amélioré par un protocole performant de tolérance aux fautes.

Des perspectives et des améliorations futures sont proposées à la fin pour raffiner ces nouvelles solutions.

Mots-clés : Systèmes répartis, Partage de ressources, Exclusion mutuelle, K-exclusion mutuelle, Tolérance aux fautes, Réseaux mobiles Ad hoc.

Table Of Contents

Table of figures

General Introduction

Nowadays, the computer represents an indispensable tool in many manipulations of our daily life, and the use of computer applications has made it possible to facilitate and accelerate the realization of various daily tasks.

This was not the case when the computer appeared in the second half of the 20th century. At that time, the computer was a slow, expensive and bulky machine that required a lot of effort to perform simple operations; and computing was only an art or a scientific prestige, and its use was very limited because of its difficulty of handling and the high costs. However, thanks to technological developments, especially in the field of electronics, the use of computers became simpler and more general, and computer science has subsequently evolved, allowing it to occupy an important place in most scientific fields.

Since the appearance of the first centralized computer system, programs needed to use both physical and logical system resources. The simultaneous access of several programs (also known as processes) to a single resource created the problem of consistency of this resource, so it is necessary to manage access to a resource in an exclusive way, i.e. only one process at a time can access to a resource. This is a well-known problem in the literature as the mutual exclusion problem, and it has been very well addressed in centralized computer systems. Several algorithms and mechanisms been proposed to solve it such as semaphores and monitors.

Computer systems evolved in a remarkable way at the end of the 20th century, so that the limitations of centralized systems could be overcome by making several remote computers communicate with each other through a communication network. This concept gave rise to distributed systems that provide access to information regardless of geographical location. These systems allow the sharing of resources between different machines, so distributed systems have inherited the problems of centralized systems, and in particular the problem of mutual exclusion.

The problem of simple mutual exclusion was treated well in distributed systems, and many solutions have been proposed. The mastering of this problem with a single resource offered the sharing of several resources between processes, this led to another problem known as K-mutual exclusion where the simultaneous access of several processes to several copies of the same resource is allowed. Particular interest was given to solve this problem in the field of distributed systems.

At the beginning of the 21st century, a very rapid growth of wireless technology and portable computing units gave birth to wireless networks; these networks are composed of mobile nodes that can be connected directly to each other or by the use of an infrastructure. But , despite this mobility which is a particularity of these networks, they represent only a special case of distributed systems, therefore, all the problems that have been dealt with in fixed distributed systems must be dealt with in mobile networks including the problem of mutual exclusion and its generalization to several resources.

Another problem that has a very negative influence on the functioning of the systems is the problem of breakdowns or faults. With this problem, the system nodes stop working in an unexpected and unpredictable way, which influences all the system, so when executing a task, the appearance of a fault will lead to the stopping of this task. The system must find a way to ensure its good functioning even in the presence of faults; in this case, we speak about fault-tolerant algorithms that can be used along with the mutual exclusion solutions to deal with different faults.

The objective of this thesis is to address the problem of K-mutual exclusion in fixed distributed systems and in mobile ad hoc networks. We expose the different categories of solutions that have been used to solve these problems and the most known algorithms in the literature, then we propose new efficient solutions to solve the problem of K-mutual exclusion, a new fault tolerance mechanism will be integrated into our algorithms to improve their performance.

The remaining of this thesis is organized as follows;

In chapter 1, we present a general view of distributed systems which is composed of a set of sites connected to each other by a communication network. The notions of these components are presented while specifying the different network topologies that may exist. After thar, we discover the mobile ad hoc networks which have taken a major interest during the last few years. The classical problems of distributed systems are presented at the end of this chapter.

Chapter 2 allows us to present in detail the problem of resource sharing in distributed systems, we understand the notion of mutual exclusion and the classes of solutions to solve this problem. We then talk about the different problems derived from simple mutual exclusion, in particular the generalization of this problem to K resources, this generalization is known as K-mutual exclusion and it is the object of our work.

Sites in a distributed system or in a mobile ad hoc network can fail, affecting the behavior of the entire system and may even lead to the shutdown of the entire system. Chapter 3 explains the concept of dependability in distributed systems and the different mechanisms that ensure this reliability, in particular fault tolerance.

In this thesis, we have presented two new fault-tolerant algorithms, the first one is presented in chapter 4, it is an algorithm that ensures mutual K-exclusion in a distributed system with a reduced number of messages for each access to a shared resource and that ensures the good functioning of the system even in the presence of several faults. A simulation whose results are well discussed has proved the performance of our algorithm.

In Chapter 5, our second algorithm is introduced, which is a fault tolerant solution to the problem of K mutual -exclusion in Mobile Ad hoc networks. In this solution, we used a new method that consists in adapting a routing algorithm to convey additional information and thus avoid high message exchange. A simulation has also proved the performance of this algorithm, and the results are well discussed.

The conclusion of this thesis proposes possible improvements and perspectives of our work.

Chapter 1:

Distributed systems and mobile ad hoc networks, an overview

Today we live in a highly computerized world, computer applications are indispensable in all fields, and they have become very greedy in computing and communication.

Centralized systems have indeed solved many problems, and they are used on a large scale, but these systems cannot satisfy the needs that are increasing day by day, so the field is open to distributed systems to take over and respond to these problems.

Networks are improving very quickly, and wireless networks have emerged, making them easier to use and set up. These networks have become the most important area of research in recent years. The evolution of wireless communication means has allowed the manipulation of information through mobile computing units, and mobile environments now offer great flexibility of use.

In this chapter, we present an overview of distributed systems and wireless networks and their applications in today's computing fields, we define also the basics about these systems that we are going to use throughout this thesis.

1. Definition of a distributed system

Several definitions of distributed systems exist in the literature, Tanenbaum for example [Tan.94] says that a distributed system is a set of independent computers that appears to a user as a single and a coherent system. Lamport [Lam.87] gave another definition, he said that a distributed system is a system that prevents you from working when a machine you have never heard about crashes. Coulouris et al. in [Cou94] define a distributed system as a set of autonomous sites connected by a communication network and equipped with software dedicated to coordinate system activities and resource sharing.

According to these definitions, we can simply define a distributed system as a set of computers, which may be geographically dispersed, and which are linked together by a communication network. Each computer is an autonomous entity capable of performing tasks independently of the others and has its local memory and its clock. The global clock and the common memory are absent, and communication between the sites is ensured only by exchanging messages via the communication network.

2. Basic elements of a distributed system

From the definition of a distributed system, we can derive two fundamental elements, the sites that make up this distributed system, and the communication network that connects these different sites.

2.1. The notion of a site

A site is an autonomous entity capable of performing tasks independently of other entities. At each site, a set of programs called processes run, and they are executed concurrently, all processes are identical and can execute the same code. For simplicity, it is assumed that there is a single process at each site, and sites have different identities.

In all that follows, the terms site, node, and process will be confused, and they will represent the same thing.

2.2. The communication network

The communication network is the physical medium that allows the exchange of messages between different sites. This network is complete, i.e. each site can communicate directly with all the other sites, but it is not practical to draw an edge between every two sites. For this reason, the communication network is represented using a logical representation that summarizes the shape of this network.

Logical representations may include the following structures:

- **The ring:** it is a circular structure, each site has two neighbors, a left neighbor, and a right neighbor, a site can only communicate directly with its two neighbors.
- **The tree:** the sites form a tree structure, where each site except the root has a father, and each site except the leaves has sons.
- **The star:** all sites communicate with one particular site called the coordinator.
- **The graph:** it is an arbitrary structure, each site can have several neighbors.

2.3 Objectives of a distributed system

The emergence of distributed systems is due to the weaknesses of centralized systems, so distributed systems must provide functions and operations that cannot be performed in centralized systems, they must allow :

- **The distribution of tasks:** a distributed system is composed of autonomous sites, thus we can exploit this characteristic to carry out complex tasks that require an important execution time by the distribution of these tasks in sub-tasks between the sites such as each sub-task is carried out on a site.

- **Resource sharing:** in a centralized system, each computer has its own logical and physical resources that are generally expensive, on the other hand, in a distributed system, several sites can use a single resource, so we can reduce the costs of use, and we can even add gradually new components.
- **Failure resistance:** since distributed systems are made up of several autonomous entities, the failure of one or many of these entities does not prevent the correct functioning of the entire system, non-faulting sites can take over the tasks intended for the failing sites.

2.4. Characteristics of a distributed system

A distributed system is characterized by :

- The different sites as a whole must have a single, comprehensive inter-process communication mechanism. This feature allows communication between different processes.
- Faulty components may restart and rejoin the system after the repair of the failure cause.
- Performance is achieved by reducing the number of transmitted messages and increasing parallelism.
- The absence of a global clock: each site has its clock, which poses a problem, how to order the events in a distributed system?

3. Ordering events in distributed systems

The absence of a global clock is one of the fundamental characteristics of distributed systems; local clocks cannot be used to date events because of the difference in clock values at each site. Nevertheless, the events produced in a distributed system must be ordered, an event can be the change of local status at a site, the sending of a message or its reception.

Local events at a site can be easily ordered using the local clock of this site. In the case of message exchange between different sites, we can find that the reception of a message sent at time ***t1*** of site ***S1,*** will take place at time ***t2*** at site ***S2***, such that ***t2*** < ***t1*** because of the difference between the local clocks. To avoid these inconsistent situations, events must be ordered even if they occurred at different sites.

3.1 Causality relation

To define an order between the different events that occur in a distributed system, several works have been done by Lamport [Lam.78] , Mattern [Mat.88], and Raynal [Ray.91a], [Ray.91b], to find a way to order the events in a distributed system.

This order is based on a relation called the causality between the events noted $\rightarrow$, and it is defined as follows:

- If *a* and *b* are two events occurring on the same site, and if *a* occurred before *b* then $a \rightarrow b$.
- If *a* is an event corresponding to the transmission of a message and *b* is the event corresponding to the reception of this message then $a \rightarrow b$.
- If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.
- Events *a* and *b* are independent or competing if neither $a \rightarrow b$ nor $b \rightarrow a$.

This relationship establishes a partial order of events because if there are two competing events, each one occurs at a different site, it becomes impossible to tell which one occurred before the other.

Let us note that events that took place in processes that do not communicate with each other form 2 disjointed sets for which it is impossible to order even partially.

To implement a global order relation in a distributed system, a new notion must be introduced: the logical clocks.

3.2. Logical clocks

We call a logical clock a function that associates each event with an integer *h(a)*, called a*'s* stamp so that if $a \rightarrow b$ then $h(a) < h(b)$, a logical clock is nothing more than a way of associating a number with an event. Indeed, this number simply represents the time at which the event occurred. It is a mechanism proposed by Lamport [Lam.78], that makes it possible to define a total order between the various events.

At each site ***i***, an integer variable called logical clock ***hi*** is used, this variable is initialized to 0, and only grows. The value of ***hi*** is incremented when an event occurs at site ***i***.

When a message is sent by site ***i***, the value of ***hi*** is incremented, and the message is stamped with (***hi***, ***i)***.

When a message is received by a site ***j,*** the value of ***hj*** will be recalculated using the function:

hj:= Max (hi, hj) +1.

This new value is the date the message was received.

As an example, we consider three processes P1, P2, and P3, whose logical clock values are initialized to 0. The following figure describes the change in the values of the stamps.

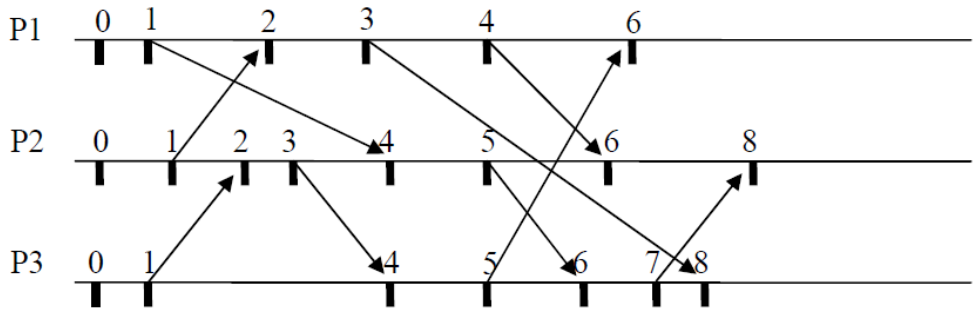


Figure 1. Logical clocks change

This order is only partial, several events can have the same values, so, to make this global order, the following function is used:

Let *a* and *b be* events on sites ***i*** and ***j*** respectively:

$$(a \rightarrow b) \Leftrightarrow (h(a) < h(b)) \text{ or } (h(a) = h(b) \text{ and } i < j)$$

By using this mechanism, all the events that occur in the distributed system can be ordered, allowing us to obtain a global order.

4. Advantages of distributed systems

- **Security:** applications are designed using a modular approach to isolate data and thus protect access.
- **Acceleration of calculations**: the distributed system allows distributing calculations to different processes to execute them simultaneously. In this way, when a site is overloaded, certain tasks can be moved or allocated to a less busy site (load distribution).
- **Transparency**: the use of resources does not affect the site status (i.e. sites use resources without knowing the location of those resources).
- **Flexibility**: Adding or removing a site is a simple operation, and does not affect the rest of the system.

- **Reliability**: One of the goals of distributed systems is to have systems that are more reliable than a centralized system. If one site fails, another site should take over, this usually results in duplication of shared data.
- **Remote access**: the same service can be used by several sites, located in different places.
- **Redundancy**: Redundant systems make it possible to compensate for a hardware fault or to choose the equivalent service with the shortest response time.

5. Mobile wireless networks

With the evolution of technology in today's world, a new mode of communication has appeared, it is the wireless communication, and mobility of sites is its strong point, thus making the implementation of networks easier without using cabling or infrastructure.

A mobile environment is a set of mobile nodes that allows users to access information regardless of their geographical location, and without using a wired infrastructure.

There are two main classes of these networks:

- Mobile networks with infrastructure: known as cellular networks.
- Mobile networks without infrastructure: known as mobile Ad hoc networks.

5.1. Mobile networks with infrastructure

Known also as the cellular networks. In this model, the network is composed of two distinct sets of entities: the fixed sites of a conventional wire-line communication network, and mobile sites. Some fixed sites, called base stations (BS), are equipped with a wireless communication interface for direct communication with sites or mobile units (MU) located in a limited geographical area, called a cell. The base stations are interconnected by a wired communication network and delimit a cell from which mobile units can transmit and receive messages.

A mobile unit can only be directly connected to one base station at any given time. It can communicate with other sites through the station to which it is directly attached. The reduced autonomy of its energy source causes it to be frequently disconnected from the network, and it can then be reconnected at a new location.

Figure 2.Wireless networks with infrastructure

5.2. Mobile networks without infrastructure (Ad hoc networks)

5.2.1. Definition of an Ad hoc network

Ad hoc networks, as a new wireless network paradigm, have many characteristics, some of which are very specific to them and differentiate them from traditional mobile networks.

- **A dynamic topology**: the mobile units of the network move in a free and arbitrary way. Therefore, the network topology can change at unpredictable moments in a fast and random way.
- **No infrastructure**: Ad hoc networks are characterized by the absence of pre-existing infrastructure and any kind of centralized administration. Mobile hosts are responsible for establishing and maintaining network connectivity.
- **Fast deployment**: Ad hoc networks can be easily installed in hard-to-wire locations, eliminating much of the work and cost typically associated with installation, and reducing the time required to get up and running.
- **Radio communication**: Communication between nodes is done using a radio interface. It is then important to adopt a medium access protocol that allows radio resources to be well distributed, avoiding collisions as much as possible, and reducing interference.

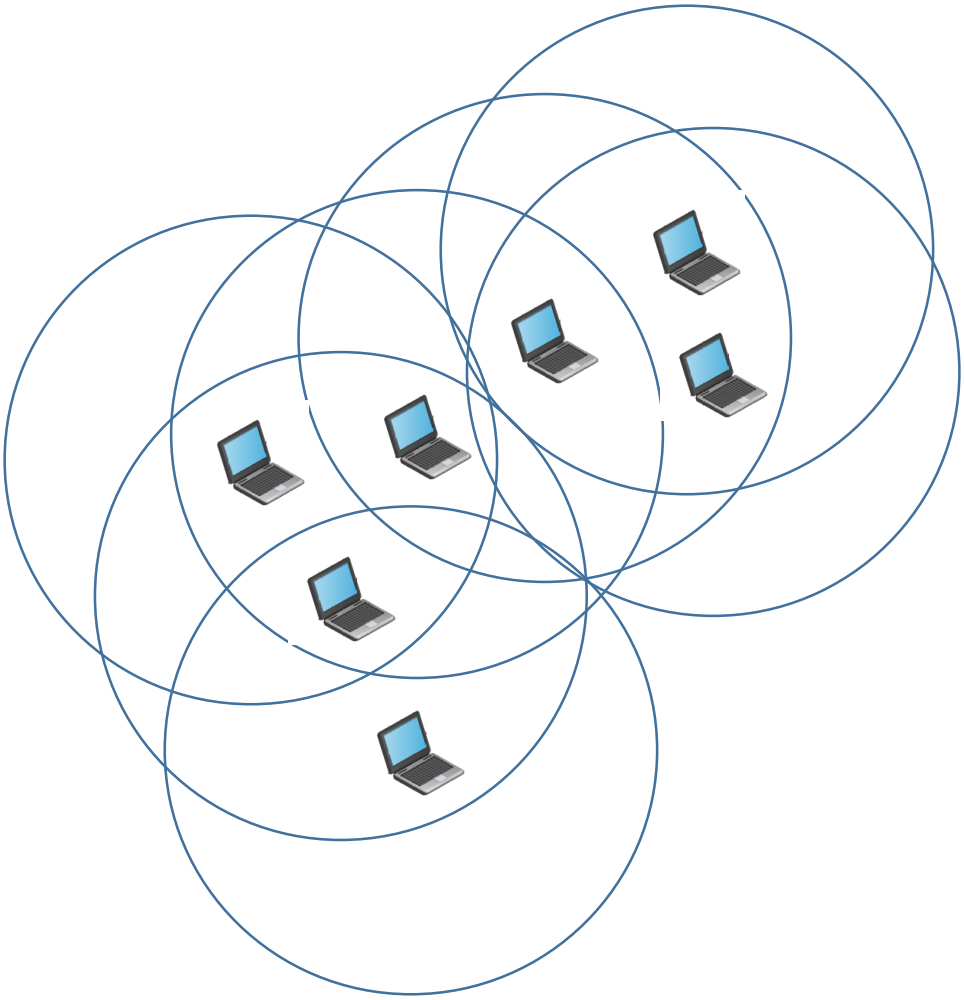


Figure 3 : Ad hoc network structure

5.2.3. Advantages of Ad hoc networks

- **Easy to deploy**: only several machines are needed to set up the network, this makes building an Ad hoc network fast and inexpensive.
- **Mobility**: the absence of cabling allows the nodes to move over time.
- **Scalable**: to add a node to a pre-existing Ad hoc network, simply approach the newcomer to at least one of the network members. Likewise, to remove a newcomer from the network, you just have to remove it from the network

5.2.4. Disadvantages of Ad hoc networks

- **Limited bandwidth**: One of the most important characteristics of networks based on wireless communication is the use of a shared communication medium, this sharing means that the bandwidth reserved for a host is modest.

- **Energy constraints**: mobile hosts are powered by autonomous and limited energy sources such as batteries, so processing time is reduced. Therefore the energy parameter must be taken into account in any control made by the system.
- **Limited security**: Mobile Ad hoc networks are more affected by the security parameter than traditional networks. This is justified among other things by the vulnerabilities of radio links to attacks, as well as the physical constraints and limitations, which means that control of the data transferred must be minimized.
- **Interference**: Radio links are not isolated; two simultaneous transmissions on the same frequency or using nearby frequencies may interfere.

5.2.5. Application Areas of Ad hoc networks

The particularity of the Ad hoc network is that it does not require any fixed installation; this allows it to be quick and easy to deploy. Tactical applications such as rescue, military, or exploration operations find in Ad hoc the ideal network. Ad hoc technology that is interested in researching civilian applications has also emerged.

Among the applications of MANETs, we can distinguish:

- **Emergency services**: search and rescue operation for people, earthquake, fires, to replace the wire-line infrastructure.
- **Collaborative work and communications in companies or buildings**: in the context of a meeting or conference for example.
- **Business applications**: for remote electronic payment or mobile Internet access.

6. Related problems to distributed systems and mobile Ad hoc networks

In a fixed or mobile environment, a site is not isolated from the others, all sites communicate with each other to perform common tasks in the case of cooperation, or to use the available resources in the case of competition. This communication leads to a set of specific problems to the distributed systems, these are classical problems that have occupied the researchers' effort for several years, among the problems of distributed systems, we can mention the following:

1. **Election**: Some distributed applications require a particular site among the sites in the system to play a particular role, this site is called the coordinator, and it ensures

communication between all the sites. The loss of communication with the coordinator site leads to the end of the application using this coordinator, the other sites must designate another site to replace the coordinator, and this new site is chosen after an election operation.

2. **Termination detection**: The sites of a distributed system can perform distributed calculations, the termination problem is the detection of the end of this distributed calculation. A calculation is said to be terminated if all the sites that participated in that calculation have completed the calculation, and there are no messages in transit between those sites.
3. **Definition of a global status**: Each process in the system has only a partial view of the system, if a problem occurs in the system, it is difficult to return to a consistent situation for all sites, so a consistent global status must be defined. The definition of this status allows a consistent recovery point to be calculated and stored, the system will return to this point when a problem occurs.
4. **Resource sharing**: Sites can use the shared resources in the distributed system, for example, several sites can use a single printer, but two different sites cannot use it simultaneously, access to critical resources must be mutually exclusive to ensure data consistency, i.e. at any given time, no more than one site can use the shared resource.
5. **Fault tolerance**: The failure of sites is an event that can unexpectedly appear in the system, these failures can have a negative influence on the system's behavior. The fault tolerance mechanisms allow the system to continue to function even in the presence of failures.

7. Conclusion

In this chapter, we have presented an overview of the fixed distributed systems and mobile Ad hoc networks.

In this thesis, we are interested in two problems in those environments, resources sharing and fault tolerance. In the next two chapters, we will detail the resource sharing problem and its relation to fault tolerance.

Chapter 2:

Resources Sharing in Distributed Systems and Mobile Ad hoc networks

In spite of all the solutions proposed to solve the various problems encountered in distributed systems, they are still topical.

In the previous chapter, the most common problems in distributed systems were generally cited as follows

In this chapter, we will present in detail the problem of mutual exclusion and its extension to K resources in distributed systems and mobile ad hoc networks.

1. The problem of mutual exclusion

1.1. The concept of mutual exclusion

Given a distributed system of ***N*** processes sharing a physical resource (a printer for example) or a logical resource (a file). Only one process at a time can use the shared resource to avoid inconsistent situations, the resource must be used in a **mutually exclusive manner** (ME). Such a resource is known as **Critical Resource** (CR), and the processes using this resource execute a section of code called the **Critical Section** (CS) that handles the shared resource.

The exclusive access to the critical resource is managed by an acquisition protocol and a release protocol. A process wishing to enter the CS must call the acquisition protocol. This protocol delays access to the CS if it is being used by another site. At the end of the CS, the process executes the release protocol to inform sites that may be on hold that the CS is available. The general form of a process using a critical resource is given as follows :

Acquisition Protocol

< Critical Section >

Release Protocol

1.2 Status of a process

Calling up acquisition and release protocols change the state of a process. Indeed, a process ***Pi*** belonging to the distributed system has a local variable ***Status*** $_i$ that designates its status, such as ***Status*** $_i$ belongs to {*Out, Requester, In CS*}.

Initially, ***Status***$_i$ is *Out*. The acquisition protocol changes ***Status***$_i$ from *Out* to *Requester* and the release protocol changes this status from *In CS* to *Out*. The change from Requester to *In CS* is defined by the system.

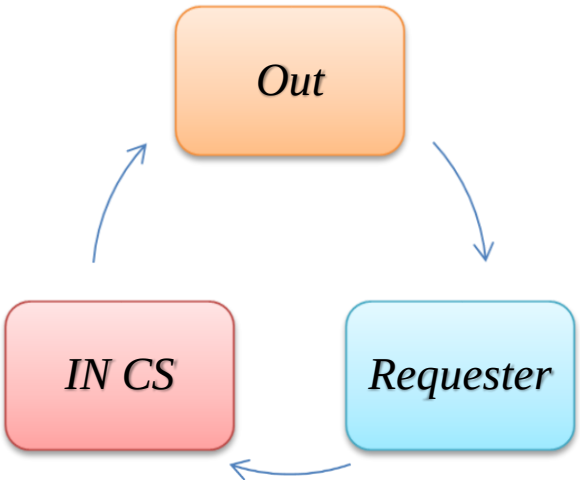


Figure 4. Different status of a site

2. Mutual exclusion solutions

To solve the problem of mutual exclusion, we use algorithms that define how to access the CR. A correct solution allows a process to pass between its different statuses while respecting the following fundamental properties:

1. **Safety property:** at any given time, there is at most one single process ***Pi*** that is using the CR.
2. **Vivacity property**: a requesting must execute the CS after a finite amount of time.

Any solution that assures these two properties, must assure also the absence of two problems, starvation and deadlock.

1. **Starvation** will occur if a process that is in the requesting status can never move to IN CS status.
2. **Deadlock** is a system situation where there are multiple processes in the requesting status and none can access the CS.

Two types of solutions can be distinguished, depending on the nature of the system, centralized solutions, and distributed solutions.

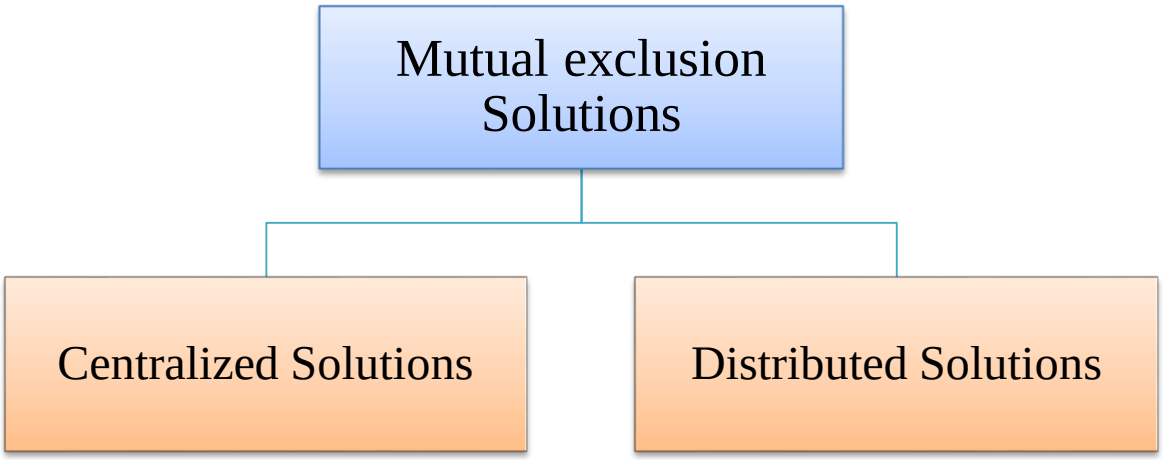


Figure 5. Solution categories

2.1. Centralized solution

Mutual exclusion algorithms are centralized when they are based on the existence of a common main memory accessible by all processes in the network, or when a particular process manages access to the critical section [Yag.97a].

This type of solution consists of designating a process known as the ***Coordinator*** to control access to the CS. The process that wishes to enter CS must request authorization from the coordinator. This latter allows only one process to access the CS, the rest of the requesting processes are put in a waiting queue. When the CS becomes available, the coordinator will authorize the site at the head of the queue to access the CS.

This solution offers several advantages, firstly, it is a simple solution, as it is sufficient to designate the coordinator to manage access to the CS for all sites, secondly, the solution is starvation free, and requests are served on a FIFO basis. However, behind these advantages, there are some disadvantages: with the use of a coordinator, there is only one point of failure; if the coordinator crashes, complex election algorithms have to be run to choose a new coordinator. Moreover, if the demand rate becomes high, the requests will occupy the communication channels, and the coordinator is overloaded to manage a very large queue.

2.2. Distributed solution

In this solution class, common memory is absent, and communication between processes is ensured only by messages exchange. Therefore, in this class, there is not a single site to manage access to the SC, but the sites must manage access to the CS in a distributed way.

Distributed solutions can be classified into two main categories according to the type of used messages, permission-based algorithms, and token-based algorithms. This classification was introduced in the work of Raynal [Ray.92], Singhal [Sin.93] and Velasquez [Vel.93].

3. Categories of Distributed solutions

3.1. Permission-based Algorithms

This type of solution consists of associating a set of sites ***Ri*** with each site ***i.*** When the site ***i*** wishes to enter SC, it must ask permission from all members of the ***Ri*** set by sending a request message. SC is executed after receiving sufficient permissions from the ***Ri*** sites.

Requests are served according to their arrival order. Since the global clock is absent, and to ensure fairness between different requests, Lamport's Stamping Mechanism [Lam.78] is used to define a global order between requests.

Permission algorithms can be further divided into three groups according to the type of permission: individual permission, referee permission, and generalized permission.

3.1.1. Individual Permission

In this type of algorithm, a site can permit several other sites simultaneously. If it is not a requester, it sends its permission to all the requesting sites, otherwise, i.e. if it is a requester, it will give its permission only to the sites whose requests have a higher priority (according to the stamping mechanism). A request is satisfied when the requesting site receives N- 1 permission.

Lamport [Lam.78] proposed the first algorithm based on this principle. When a site wants to enter SC, it sends a request to all other sites, the site with the oldest request can enter SC, the number of messages used is 3(***n-1***) where ***n*** is the number of sites in the system.

Ricart and Agrawala [Ric.81] have proposed an improvement of this algorithm; a site wishing to enter SC must send requests to all other sites in the system, and wait for their permissions. 2(***n-1***) messages are sufficient for each SC entry. In these two algorithms [Lam.78], [Ric.81], the requesting sites send their requests to the same sites each time, i.e., the ***Ri*** sets are static.

To reduce the number of exchanged messages, another algorithm has been proposed by Carvalho and Roucairol [Car.83], in this algorithm, the requesting site sends its requests only to sites that have entered SC since its last entry, which makes the ***Ri*** sets dynamic, therefore the number of messages is reduced, and varies between 0 and 2(***n-1***).

3.1.2. Referee Permission

In this case, a site ***i*** wishing to enter SC sends its request to a sub-set of all the sites ***Ei***. At a given moment, a site with several requests grants its authorization to only one of them, the others being put on hold. Once the CS is executed, the site must return the permissions that have been granted to it so that permission can be given to the pending requests. Maekawa [Mae.85] proposed the first algorithm based on this principle: the sites are divided into subsets so that the intersection between two subsets is not empty. This ensures mutual exclusion, and dividing the sites into groups reduces the number of exchanged messages. However, the fact that a referee permits only one requester can lead to deadlock situations. Moreover, the construction of the sets is very complex.

3.1.3. Generalized permission

These algorithms combine the two strategies described above. They were introduced by Sanders [San. 87] and were taken up by Singhal [Sin92]. Thus, each site ***i*** maintains a subset of sites ***Ei***. When a request is received from a site ***j*** belonging to ***Ei***, it will give a referee type permission, and if not (***j*** does not belong to ***Ei)*** individual type permission will be sent. These algorithms solve the deadlock problem of Maekawa's algorithm.

3.2. Token based algorithms

The principle of these algorithms is to use a particular message called a token, and only the site holding this token is authorized to execute its SC. It is clear that the uniqueness of the token ensures the ME, the problem lies in the way the token is circulated between sites, in other words, how to allow a requesting site to have the token to access the CS?

We can distinguish two types of token-based algorithms according to the way the token circulates, algorithms based on diffusion, and those based on the use of a logical structure.

3.2.1. Algorithms based on diffusion

In these algorithms, a process that wants to request CS can send its request in parallel to all the other sites and wait for the reception of the token, these algorithms can be static or dynamic.

1. **Static algorithms:** a site that needs the token to enter SC sends its request to all the other sites; on receipt of this token, it accesses SC. Once out of its critical section, it sends the token to the oldest site whose request has not yet been satisfied. Suzuki-Kasami's algorithm [SKA.85] is an example of algorithms using this mechanism.
2. **Dynamic algorithms:** To reduce the number of messages and to avoid soliciting sites that do not use the critical section often, sites maintain a history of the last requesters. A request is sent then only to the present sites in history. Chang et al [CSL.91] proposed such an algorithm in 1991 by improving the Suzuki-Kasami algorithm [SKA.85].

3.2.2. Algorithms based on a logical structure

In this group, the sites form a logical structure and the token is sent via this logical structure. The logical structure can remain unchanged during the execution of the algorithm, so we speak of a static structure, or it can change during execution, in which case we speak of a dynamic structure.

Static algorithms :

Static algorithms use fixed structures that do not evolve with the system. In Lelann's algorithms [Lel.77] and [Lel.78] the token circulates along a unidirectional ring. Since the token circulates along the ring, a process will certainly have the token at some point in time.

The tree is another static structure used, in the algorithm of Raymond [Ray.89] or Nelsen and Mizuno [Nel. 91] for example, the root node of the tree is the site that holds the token. The links are oriented towards the root so that when a site requests the token, this request is propagated to the root.

The sites can also be structured in an arbitrary structure, in the algorithm of Helary et al [HPR.88] or the algorithm of Chang et al [CSL. 91], the structure used is a graph.

Dynamic algorithms

The logical configuration adopted to the network changes when a process requests and executes the CS. This dynamic logical structure allows the execution history of the critical resource to be traced. Knowing who holds the token or who is going to hold it allows us to gain a lot in the number of exchanged messages and therefore improve the performance of the algorithm. The algorithms of Naimi, Trehel [NaT.87a], [NaT.87b] and [NaT.96] are the references in this category.

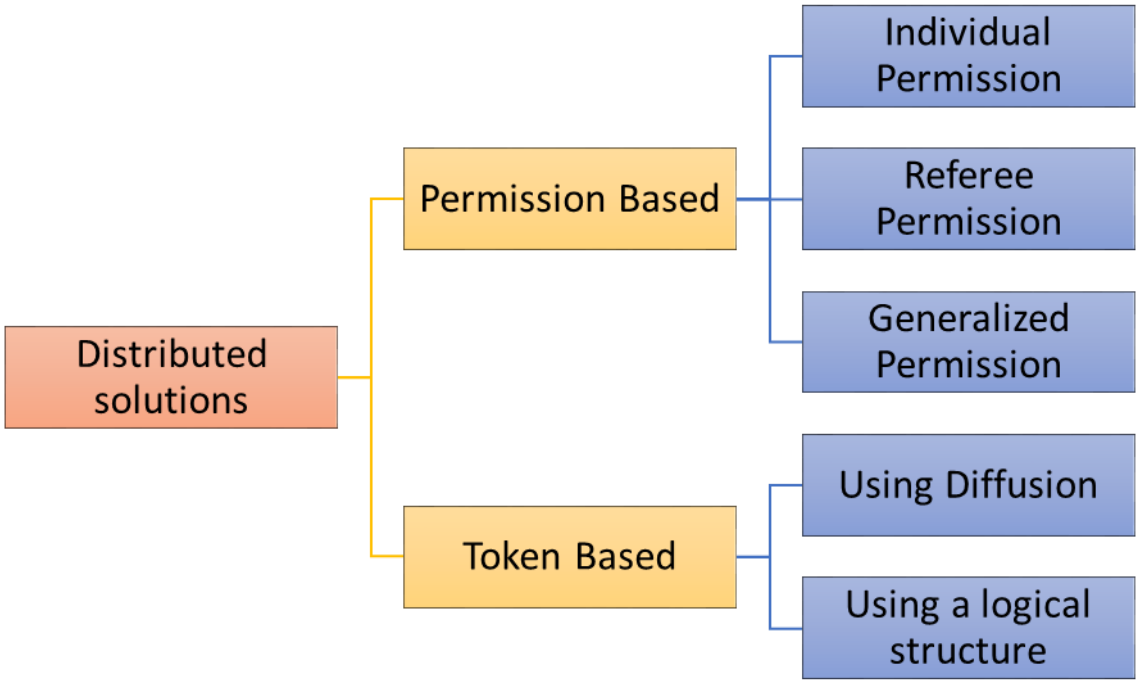


Figure 6. The Categories of Distributed solutions

4. Extensions of mutual exclusion

Since the problem of mutual exclusion first appeared in the literature, several solutions have been proposed, and they have made it possible to ensure that different sites access to the critical resource in the order in which requests are received. However, several extensions of this classic problem have been addressed.

4.1 Mutual exclusion with priority

In this extension, requests are associated with a priority level, and the goal is to serve requests according to the priority level and not on a FIFO basis. Several algorithms have been proposed. such as Housni and Trehel's algorithm [Hos.01].

4.2 Time sensitive mutual exclusion

In this case, the requests must be served before the due date, the scope of this extension is usually in real-time systems.

4.3 Group mutual exclusion

A group of processes in the system can simultaneously access a critical resource, this notion was introduced by Joung [Jou.98] and known in the literature as the philosophers' conversation, where philosophers spend their time thinking alone or discussing a particular topic together in a meeting room. When they stop thinking (wish to enter the critical section), they choose a topic of conversation of their choice to discuss it in the meeting room. Since there is only one room (the shared resource), the conversation can begin if and only if the room is empty. Philosophers interested in the discussion can join it along the way. Conversation time is assumed to be limited to ensure that the room will eventually be available for another conversation. The security interest ensures that there is no more than one topic of conversation in the room. The liveliness property ensures that any philosopher will access the room in time to finish with the topic of conversation that interests him. Finally, we can add a competition property ensuring that any philosopher interested in a conversation in progress will be able to access the meeting room.

4.4 h out k mutual exclusion

In this case, ***k*** copies of a single resource are available, and each process may request ***h*** resources at a time, but it must be guaranteed that only one process uses a copy of the resource at any given time (*safety property*), at the end of the CS, the site must release all used copies. This type of allocation is widely used in multimedia networks, for example, a site that wants to broadcast a video sequence must reserve several communication channels among the channels available to transmit the video and sound, the release of the channels is made at the end of the broadcast (release of all the resources at once).

Several algorithms have been proposed to solve this problem, such as the work of Baldoni et al [Bal. 95], and the Jiang algorithm [Jia. 03].

4.5 K-Mutual Exclusion

In this case, the system has several copies of the same critical resource, so several processes can be in a critical section simultaneously.

This extension is the subject of our work, so a detailed study is presented in the following section.

5. The problem of K mutual exclusion

5.1. Description of the problem

This is a problem derived from the classical mutual exclusion and known as the K-mutual exclusion problem (KME) or K-mutex problem; the difference between the two problems lies in the number of copies available for a shared resource. Instead of a single resource as in ME, there are ***K*** copies of the same critical resource, but this multitude of resources does not allow simultaneous access to the same resource, i.e. a process can only use one copy at a time, and a resource can be allocated to only one process.

To solve this problem, the main properties of a mutual exclusion solution remain valid, therefore it is necessary to ensure:

- ***Safety***: exclusive access to a resource must be ensured.
- ***Vivacity***: any request must be satisfied after a finite amount of time.

5.2. Solving the problem

To solve the KME problem, several approaches can be used. The first approach uses a simple idea based on the fact that the KME problem can be reduced to the ME problem, this idea consists in managing access to each resource independently of the others using a correct solution of mutual exclusion among those proposed. To do this, each copy of the critical resource has its own identity, and the requesting process must specify the identity of the requested resource at the time of its request.

The advantage of this solution is its simplicity; we just have to choose a solution with an acceptable complexity. But, the major problem of this solution is the long waiting time; this waiting time will occur if we have a large number of requests for the same resource, as the safety property allows only one process to use the resource at a time, the others have to wait for its release; while other free resources can be found, so we must find a way to manage free resources. This idea based on resource identification is not practical.

A second idea is to solve the KME problem using the basis used to solve the classical mutual exclusion, so there are two types of solutions:

1. **Permission-based solutions:** the process wishing to enter the CS must request permissions from a set of processes; the receipt of a sufficient number of these permissions authorizes the process to use the resource.
2. **Token-based solutions:** in this case, there is not a single token but ***K*** tokens (***K*** is the number of resources), only the possession of a token allows a requesting site to access the CS.

5.3. Related works

Raymond [Ray.89] proposed the first solution of KME, this algorithm is based on the algorithm of Ricart and Agrawala [Ric. 81]. The algorithm of Srimani and Reddy [Sri.92] was then proposed, and it was an improvement of that of Suzuki and Kasami's algorithm [Ska.85], Kakugawa et al. used in [Kak.94] a close method to the method of Maekawa [Mae.85], Naimi also presented a generalization of his ME algorithm [Nai. 87] to solve the K-EM problem in [Nai.93].

These algorithms improved the simple ME algorithms to ensure KME. However, algorithms are using original ideas such as the algorithm of Bulgannawar and Vaidya [Bul. 95] and the algorithm proposed by Baldoni and Ciciani [Bci. 94].

6. Resources sharing in mobile Ad hoc networks

In a mobile environment, mobile sites can share the available resources in the system, while respecting mutual exclusion.

Like the fixed distributed systems, and since the proposition of the first solutions by Walter et al. in [Wal.97] and [Wal. 98], several solutions have been proposed to solve this problem in mobile Ad hoc networks by taking into consideration the particularity if such an environment. In this section, we present the different categories of ME solutions.

6.1. Cluster-based approaches

To overcome message complexity, clustering-based architectures are the main existing solutions. Same as all MUTEX solutions, all existing solutions falling under this category are, to the best of our knowledge, token-based solutions.

A cluster-based K-MUTEX algorithm for MANETs is proposed in Haghighat and Mohamadi [Hag.11]. In this algorithm, the network must be divided into at least K clusters. Based on the Raymond algorithm [Ray.89], nodes within the cluster get the role of head, gateway, or ordinary node, where every cluster head maintains the list of requesting nodes. The cluster head attempts to serve demands using the cluster's token, but if there is no available token, it forwards the request to the other cluster heads. This technique was proposed to minimize the number of exchanged messages. However, the cluster head election may become a serious problem, especially in mobile environments.

In [Erc.04], the author proposes an architecture consisting of a ring of clusters for distributed mutual exclusion algorithms in the scope of mobile networks. They periodically re-divide the network into new clusters based on the fixed centered partitioning algorithm, and then every cluster implements a separate MUTEX algorithm. They applied Ricart-Agrawala [Ric.81] and token-based algorithms to evaluate the performance of their architecture. Despite simulation results show that message complexity is considerably reduced, this solution cannot be considered as an adequate option for solving the K-MUTEX problem over MANETs.

Same as Erciyes [Erc.04], Dagdeviren and Erciyes [Dag.07] propose an architecture for MUTEX handling in MANETs. Specifically, they divide their software architecture into three layers. At the lowest layer, the merging clustering algorithm (MCA) periodically partitions the MANET into several balanced clusters. At the second layer, the backbone formation algorithm (BFA) provides a virtual ring using the cluster heads found by MCA. The final layer is represented by the implementation of a modified and scaled version of the Ricart-Agrawala algorithm. Unfortunately, this architecture inherits the previously mentioned problems.

6.2 Fully distributed approaches

Solutions falling in this category do not make any clustering assumptions, and their communications are fully distributed. Generally, this category's solutions suffer from the generation of an excessive number of messages.

6.2.1 Token-based approaches

A token-based distributed algorithm for supporting MUTEX in opportunistic networks is proposed by Tamhane and Kumar in [Tam.12]. This approach tries to achieve a trade-off between the dynamic MANET topology and the token-passing process, and it can be divided into three main procedures:

- Request generation
- Request propagation
- Token propagation.

Every node in a critical section (CS) case maintains the list of demanding nodes, and then it releases the Token for the first demander and acts as an intermediate for the others. Even if this approach can guarantee that all demanders will be served, it cannot handle link or node failures. The work of Thiare and Naimi [Thi.09] presents a group K-MUTEX algorithm for MANETs where authors assume that there is a unique token initially, and then utilize the partial reversal technique to maintain a token-oriented DAG (directed acyclic graph). Then, a node

holding the token can enter the CS, and inform all its requesting neighbors by sending out Okay messages. When a neighbor node receives the Okay message, it can enter the CS if, and only if, it requests for the same resource as the token holder. Additional information messages and the continuous DAG computation are the main problems of this technique. Furthermore; there are no complexity studies nor simulation tests that validate this approach.

In another work, Walter et al. [Wal.01] propose a DAG-based k-resources sharing technique for MANETs where the token holders are considered as *sink-points*. Besides, every node maintains a queue of requesting node identities. The problems with this solution are mainly:

1. The assumption that the token is reliable and will never disappear;
2. That duplicates do not occur;
3. That node failure does not occur.

6.2.2 Permission-based approaches:

Due to the unpredictable and dynamic topology of MANETs, most of the existing works adopt token passing strategies instead of permission gathering. Hence, only a few works fall under this category.

A permissions-based technique for solving the K-MUTEX problem is proposed by Masum et al. [Mas.10]. The authors of this technique tried to overcome mobility and disconnection problems by gathering enough consensuses for every mobile node intending to enter the CS. They propose creating a specific message containing the information required to reach such consensus, including ProcessID, LastRequestID, and LastRequestTS (TimeStamp), among others. Unfortunately, this approach does not take into account MANET's delay sensitivity, nor the additional messages overhead or the energy consumption associated with the transmission process.

7. Conclusion

In this chapter, we have presented a general view on the problem of mutual exclusion in distributed systems and mobile ad hoc networks with their extension to K resources, the proposed algorithms allow solving this problem, but they still need improvement concerning the number of exchanged messages and also the treatment of site failures.

In the following chapter, we will study the operational reliability of computer systems, which is a very important concept that allows systems to function properly despite the presence of failures.

Chapter 3:
Dependability and fault tolerance

The performance of the algorithms proposed to solve the various problems encountered in distributed systems remains limited if they do not take into account an important event which is the failure of the sites, such an event can cause problems and affect the proper functioning of the system, for this we must ensure the safety of operation in the systems.

In this chapter, we will introduce the notion of dependability and the means used to ensure it, and in particular fault tolerance.

1. Concept of dependability

The dependability of the computer system is the property that enables its users to place justified confidence in the service it provides. Reliability is concerned with improving the survivability of the system, i.e. its ability to continue to function during and after a disruption encountered during its operational life [Lap.92].

2. Attributes of dependability

The attributes of the dependability focus more or less on the properties that the safe operation of the system should verify. These attributes are used to assess the quality of service provided by a system.

1. **Availability**: This is the property required by most safe operating systems. It is the ability of a system to provide a service at any time, and to perform its functions without interruption and to respond to a demand at a specific time.
2. **Reliability**: This attribute evaluates the continuity of service, i.e., the system is in a continuous state of service.
3. **Safety**: similar to reliability but concerning catastrophic consequences caused by faults, i.e. a system malfunction does not have a catastrophic impact on its environment.
4. **Maintainability**: This property describes the flexibility of the system to changes made for maintenance purposes, i.e., the ability of the system to be repaired or upgraded and its ability to restore proper service after a failure

3. Impediments to dependability

A system fails (system failure) when the service it delivers deviates from the expected service. Therefore, dependability seeks to avoid failures, or at least to prevent the most catastrophic ones.

Failure occurs because the system behaves incorrectly: an error is the part of the system state that is likely to lead to failure. The adjudicated or assumed cause of the error is a fault. An error is, therefore, the manifestation of a fault in the system, and failure is the effect of an error on the service.

3.1. Faults:

A fault or breakdown is the cause of a failure, it is characterized by its nature, its origin, and its temporal extent. The nature of a fault specifies how it was caused: intentionally or accidentally. The cause of the occurrence of a fault is revealed by its origin. The temporal scope characterizes the persistence or duration of a fault (temporary or permanent).

Faults can be classified as follows:

Classification of faults according to their nature :

- **Accidental faults**: faults occur accidentally.
- **Intentional faults**: they are created with an intention that may be malicious.

Classification of faults according to the permanence degree

- **Transient fault**: it occurs in isolation;
- **Intermittent fault**: occurs randomly several times;
- **Permanent fault**: it persists as soon as it appears until repaired.

Classification of faults according to the seriousness degree

- **Crash fault**: either the system is working normally (the results are correct) or it does nothing. This is the simplest failure model, which we try to get back to whenever possible;
- **Faults by omission**: messages are lost on entry and/or exit. This type of fault is used for networks;
- **Timing faults**: Deviations in regards to the specifications are only related to time (typically the reaction time to an event) ;
- **Value faults**: the results produced by a faulty component are incorrect;
- **Byzantine faults**: the system can behave in a very arbitrary way (can do anything, including malicious behavior).

3.2. Errors:

An error is the consequence of a fault. A fault occurs as soon as the system uses an erroneous state. Firstly, the service does not correspond in value to the one specified. Second, the service is not delivered within the specified time interval. The service provided is always early or always late or arbitrarily early or late. The fact that the system fails to render the service

is also considered as a type of error. A crash is defined by the fact that the system permanently fails to deliver services.

3.3. Failures:

A failure indicates the inability of an element in the system to provide the service specified by the user. Its domain, its perception by users, and its consequences on the environment characterize the failure.

4. Means of ensuring dependability

The means of dependability are the methods, tools, and solutions that provide the system with the capability to provide a service following the specified service and give confidence in that capability. These means make it possible to break the Fault, Error and Failure sequences and thus improve system reliability.

The design and implementation of a secure computer system require the combined use of a set of methods that can be classified as follows:

4.1. Fault prevention

Fault prevention is based on methods development and techniques implementation to prevent faults from being introduced during the development phase, i.e. how to prevent, by construction, the occurrence or introduction of faults. Fault prevention focuses on the means to prevent the occurrence of faults in the system. These are generally the approaches to checking conceptual models.

4.2. Fault elimination

Fault elimination can be divided into 2 categories: elimination during the development phase and elimination during the use phase. During the development phase, the idea is to use advanced checking techniques to detect faults and remove them before sending them to production. During use, faults encountered must be kept up to date and removed during maintenance cycles.

4.3. Fault prediction

Consists of estimating, by evaluation, the presence and creation of faults and their consequences. Fault prediction predicts the occurrence of faults (time, number, impact) and

their consequences. This is generally done by fault injection methods to validate the system relative to these faults.

4.4. Fault tolerance

Fault tolerance tries to work despite the faults. The degree of fault tolerance is measured by the ability of the system to continue to deliver its service in the presence of faults.

5. Fault tolerance techniques in distributed systems

Fault tolerance in distributed systems is always achieved using a redundancy mechanism. This mechanism can be spatial (duplication of components), temporal (multiple processing), or informational (data redundancy, codes). In such a system based on communicating processes, fault tolerance techniques can be separated into two classes: techniques based on stable memory and techniques based on duplication

5.1. Fault tolerance using a stable memory

The existence of a stable memory allows the fault tolerance to be achieved through fault detection followed by error recovery. Stable memory can be defined as a persistent storage medium; the main role of that storage is to ensure accessibility and protection of data against faults that may affect the system. Thus, when a fault occurs, a correct state having been stored before this fault on the stable memory remains accessible; this allows the system to return to a previous state.

Once a fault is detected, a recovery mechanism must be put in place to deal with the identified fault. As already noted, to be able to perform a recovery from a fault, the state of the system or application must be saved periodically or upon the occurrence of specific events. The saved state must be a recovery point from which the system can function properly.

5.2. Fault tolerance by duplication

Duplication fault tolerance is the creation of multiple copies of components on different processors. This duplication approach makes it possible to handle faults by hiding them. Three main strategies for replication are proposed in the literature: active, passive, and semi-active replication. These different strategies aimed at ensuring a strong coherence between the copies of a duplicated component.

5.2.1 Active replication (N-modules replication)

It is defined by the symmetry of the behavior of the duplicated copies of a component where each copy plays an identical role to the others.

5.2.2 Passive duplication (primary-backup replication)

Unlike active replication, passive replication is asymmetrical. It distinguishes between two behaviors for the duplicated copies of a component: primary copy and secondary copies (backups). The primary copy is the only one to perform all processing. The secondary copies, idle, monitor the primary copy. If the primary copy fails, one of the secondary copies becomes the new primary copy.

5.2.3 Semi-active duplication (root-follower replication)

Semi-active replication is halfway between active and passive replication. Like passive replication, semi-active replication is an asymmetric strategy. Unlike passive replication, secondary copies are not idle.

Duplication fault tolerance is then achieved by error concealment. The failure of a copy is masked by the behavior of non-failed copies. This technique tolerates a limited number of faults and is not suitable for massively parallel computations in terms of performance and computational resources required for its implementation.

6. Related works

Several algorithms that tolerate faults in different systems exist, but we are interested in the fault tolerance in the mutual exclusion and K mutual exclusion algorithms in distributed systems and mobile ad hoc networks.

For static structures, Lelann proposed a fault tolerance mechanism for his algorithm that uses the ring structure [Lel.78]. Raymond proposed another algorithm for the tree structure [Ray.89].

For algorithms with a dynamic structure, we find the Naimi-Tréhel algorithm [NT87b] which is based on the structure of a dynamic tree, Agrawal and El Abbadi have proposed another algorithm [Agr.91] which is based on the same structure.

The algorithm of Helary and Mostefaoui [Hel.94] exploits the symmetry properties of the open-cube to tolerate outright failures.

Yagoubi et al. [Yag.97b] proposed a fault-tolerant algorithm for real-time systems based on duplication.

For the mobile Ad hoc networks, several algorithms have been proposed. Wu et al. [Wu.07] propose the mobile dual-token ME (MDTM) algorithm for detection of token loss; the algorithm detects token loss due to links problems only by using two types of tokens, primary and secondary. Each token has its dynamic ring and a coordinator. Only the primary token grants permission to enter the CS. If a token completes a cycle over the ring and does not find the ID of another token on any of the nodes, it assumes that the latter is lost and must be regenerated. This approach is not suitable for MANETs because of the ring assumption. The same authors have proposed another permission-based fault-tolerant solution in [Wu.08]. It is based on a look-ahead technique similar to medium access techniques. In particular, this technique enforces ME only among the hosts currently competing for the critical section, thus reducing the number of exchanged messages. Besides, they use a predefined timeout to detect mobile hosts' disconnection. Despite supporting high levels of mobility, this permissions-based approach introduces significant delay problems.

Moallemi et al. [MOA.07] proposed another fault-tolerant K-ME technique for MANETs. This solution extends the well-known Naimi-Trehel's algorithm [Nai.87] with an entropy-based clustering algorithm where cluster heads manage the token passing among requesting nodes. However, the authors fail to provide experimental results or analytical proofs of this solution. Hence, the actual performance of this technique remains unknown.

7. Conclusion

In this chapter, we introduced the concept of dependability and fault tolerance in distributed systems and mobile ad hoc networks.

Then, we identified the different algorithms proposed to ensure fault tolerance, as we have noticed; most of these algorithms are token-based algorithms.

In the following chapters, we will present our contributions, in chapter 4, we present a fault-tolerant algorithm to solve the K-ME problem in distributed systems, and then, in chapter 5, we introduce a fault-tolerant algorithm in Mobile Ad hoc networks.

Chapter 4:
A new fault-tolerant token-based K-ME algorithm in distributed systems

Several algorithms have been proposed to solve the ME and the KME problems, and these problems are still relevant today. Nevertheless, nothing is perfect, and nothing is complete, all the proposed solutions are indeed correct, and each new solution may offer more acceptable performance, but these solutions still need improvement, especially in terms of complexity in the number of messages. We also note that the KME problem has not received as much attention as the simple mutual exclusion problem. This is why we have thought of proposing a new solution to solve the KME with a much-reduced number of messages.

In this chapter, this new solution will be presented in detail with its proof of performance, and we will show that its complexity is better than those of the algorithms studied in this field. Also, a fault tolerance mechanism will be integrated into our proposal to make it more efficient.

1. Motivation

In a distributed system, sites communicate with each other by exchanging messages to carry out distributed cooperative or competitive tasks. It is clear that the number of exchanged messages has a significant influence on the performance of the system, the more the number of messages increases, the more the performance of the system decreases, therefore the number of exchanged messages needs to be minimized to ensure an acceptable performance level.

The allocation of shared resources is not an exception; each algorithm requires several messages to ensure exclusive access to a critical resource.

The KME algorithms look at two important values, the value of ***N***, which represents the number of sites in the system, and the value of ***K***, which is the number of shared resources between sites. Usually, the value of ***N*** is much higher than ***K,*** it is quite logical, many nodes are competing to acquire a limited number of resources.

The study of the KME algorithms already presented in chapter 2 shows that the complexity of these algorithms is expressed as a function of ***N***, for example, Raymond's algorithm [Ray.89] requires $2NK+1$ for each SC entry, Naimi's algorithm [Nai.93] requires $2(N-1)$ messages, and the list is long. Since this complexity is expressed as a function of ***N*** it will be very high, as the value of ***N*** is very large, it will overload the system with exchanged messages at each access to a resource, adversely affecting the system performance.

Therefore, It will be very useful and more interesting to express the complexity in terms of a value lower than ***N.*** If we can propose a solution with a complexity expressed in terms of

K for example instead of ***N***, we will have an interesting improvement in the exchange of messages, which will have a positive influence overall the system.

2. The basic idea

In a KME algorithm, the complexity of the messages depends on the logical structure used by the algorithm, and the number of sites participating during the acquisition of the CS. If the number of links a request traverses is high, the number of messages becomes very large (as in the Naimi's algorithm [Nai. 93] where the request can traverse the entire network). Moreover, if the access to the CS requires the permission of all the sites of the system, the number of messages becomes also very important (case of Raymond's algorithm [Ray.89]). Therefore, using a simple logical structure, or not requiring that all sites be involved in the acquisition of the CS may reduce the number of exchanged messages.

Previous works such as the algorithm of Maekawa [Mae. 85], or Kakugawa et al [Kak. 94] show that the decomposition of sites into groups allows obtaining a reduced complexity. Dividing the ***N*** sites into groups will give sub-sets whose cardinal is lower than ***N,*** thus the number of messages becomes according to this cardinal or according to the number of groups instead of ***N***, allowing to reduce the complexity.

Based on this last principle, and to be able to express the complexity as a function of ***K,*** we propose to divide the ***N*** sites into ***K*** linked groups. This decomposition allows us to reduce the number of messages and to render them as a function of ***K*** instead of ***N***.

This decomposition should not influence the connectivity of the sites, i.e. the network must not be partitioned, so we will combine two logical structures to ensure that all sites are connected.

The first structure is a logical tree, in each tree, a particular site is designated to play the role of the root, and a bidirectional logical ring, which represents the second logical structure, will link the roots of all trees together, this ring allows the exchange of messages between all the root sites. This choice was not random, our algorithm combines these two structures thanks to their simplicity on the one hand, and to exploit the advantages offered by each of them on the other hand, for example, in a tree, the distance between each site and its root is one, which makes it possible to minimize the number of exchanged messages.

To ensure KME, we use as many tokens as available resources, initially, each root holds one free token and the roots can exchange tokens with each other, a root can hold up to ***K*** free tokens at a time.

When a requesting site wants to enter CS, it sends a request to the root of its tree, this root will satisfy the request by sending a token if it holds free tokens, if not, the request will be saved, and the root will request an additional token from its neighbors via the ring.

If the root that receives the request from a neighbor holds free tokens, it will respond favorably to this request by sending a token, otherwise, it will propagate this request to another neighbor.

The reception of a token by a root serves the first pending request, so the requesting site that receives a token passes to the CS, and sends the token to its root after the CS is released.

To avoid the costly exchange of tokens between the roots, we use a judicious method that allows us to send the tokens via the shortest path between the sending site and the receiving site, thus minimizing the number of exchanged messages for each CS entry.

3. The logical structure

The idea of our algorithm is to divide the sites into groups. But, how to define these groups?

There are several methods for dividing sites into groups, we can even obtain the groups in a random way, but we risk having overloaded groups, which directly affects the fairness of our algorithm.

For reasons of simplicity, and to ensure that all groups are well balanced, we gather the site into groups according to their identities; our method is to form the groups in two steps to obtain a logical structure consisting of static trees and a bidirectional ring.

Given a system composed of ***N*** sites numbered from 1 to ***N***. The first step allows creating static trees such that, for $1 \leq i \leq K$, *the* site ***i*** is the root of a tree noted $Tree_i$, and for the rest of the sites*: K+1, K+2,...,* ***N,*** each site is added to a tree according to its identity using the following method:

If $i \equiv 1 \bmod K$*, this site belongs to* $Tree_1$

If $i \equiv 2 \bmod K$*, this site belongs to* $Tree_2$

If $i \equiv 3 \bmod K$*, this site belongs to* $Tree_3$

. . .

If ***i*** ≡ *K-1 mod K, this site belongs to* $Tree_{K-1}$

If ***i*** ≡ *0 mod K, this site belongs to* $Tree_K$

These trees have a static structure, and each site is linked directly to its root.

The second step consists in connecting the trees, for this purpose, only the ***K*** roots will be connected by a bidirectional ring such that, for a site ***i,*** the left neighbor of this site is the site ***i-1,*** and the right neighbor is the site ***i+1***. (For sites 1 and **K**, we have: left neighbor of site 1 is site ***K,*** and right neighbor of site ***K*** is site 1).

4. Proposal details

In this section, we present the local variables, the used messages, and the algorithm procedures, some explanations are associated with the procedures to facilitate the understanding of the code.

4.1. Data design

The local variables are not the same because there are two types of sites, roots, and regular sites:

At a regular site *i*

- ***Status*:** Refers to the status of the site, this status can be Out, Requesting, or In CS.
- ***Root*:** The identity of the tree root containing this site.

At a root *i*

- ***Status*:** Refers to the status of the site, this status can be Out, Requesting, or In CS.
- ***Root*:** nil
- ***Left_Neighbor:*** the identity of the left neighbor in the ring.
- ***Right_Neighbor:*** the identity of the right neighbor in the ring.
- ***Free_Tokens:*** an integer that represents the number of free tokens held by the root.
- ***Tokens_present:*** an integer that represents the number of tokens in the tree (free and used).
- ***Requesters:*** a queue to store requests.
- ***Length:*** the length of the queue.
- ***Distance:*** a variable to calculate the distance between two sites in the ring.

4.2. Initialization

At a regular site *i* belonging to $Tree_j$:

- ***Status=Out***.
- ***Root= j*** .

At a root *i*

- ***Status=Out***.
- ***Root*** **=** nil.
- ***Free_Tokens=1***.
- ***Present_Tokens=1***.
- ***Requesters*** =nil.
- ***Length*** =0.
- ***Distance*** =0.

4.3. Proposal used messages

In this algorithm, six types of messages are used:

- **Request (Token, *i)*:** Sent by site ***i*** to its root when requesting CS.
- **Release (Token):** Sent by site ***i*** to its root after the release of its SC.
- **Agreement (Token):** Sent by a root to a requesting site to authorize it to use the requested critical resource.
- **Request (Token, *i)*:** Sent by a root ***i*** to a neighbor in the tree to request an additional token.
- **Help (Token, *i)*:** Favorable response to a neighbor's request, this message is sent from one root to another.

4.4. Proposal procedures

Procedure1: Request CS

```
1:    Begin
2:        Status := Requesting;
3:        Send Request (Token, i) to Root;
4:    End;
```

Procedure2: Receipt of Request (Token, *j*)

```
1:    Begin
2:        If Free_Tokens > 0 then
3:            Send Agreement (Token) to j;
4:            Free_Tokens:= Free_Tokens -1
5:        Else
6:            If Present_Tokens< Length then
7:                Send Request (Token, i) to Right_Neighbour
8:            End If
9:            Requesters:= Requesters + {j}
10:           Length:= Length+1
11:       End If
12:   End
```

Procedure 3: Receipt of Request (Token, j)

```
1:    Begin
2 :       If Free_Tokens > 0 then
3 :           Send Agreement (Token) to j;
4 :           Free_Tokens:= Free_Tokens - 1;
5 :       else
6 :           If Present_Tokens< Lengthi then
7 :               Send Request (Token, i) to right_neighbor ;
8 :           else
9 :               Send Search (Token, i) to right_neighbor ;
10 :          end if ;
11            Requester := Requester + {j};
12 :          Length:= Lengthi + 1;
13 :      End if;
14 :  End;
```

Procedure 4: Receipt of Search (Token, j)

```
1 :  Begin
2 :      If Free_Tokens_libres > 0 then
3 :          Use the shortest path (j);
4 :          Free_Tokens:=Free_Tokens - 1;
5:           Present_Tokens:= Presents_Tokens - 1;
6 :      Else
7 :          Send Search (Token, j) to right_neighbor ;
8 :      End If;
9 :  End;
```

The root that receives a request from a site in the tree will grant this request by sending a token if it holds free tokens. Otherwise, there are two situations: if the number of the present tokens used in the tree allows satisfying the requests after a finite time, the root will wait for the release of a token. If not, the root will request an additional token from a neighbor in the ring by sending a request message, in both cases; the request will be added to the queue. The Search message aims at minimizing the waiting time, instead of waiting for the release of a token in this tree, we can search for a possible free token at the other roots to minimize the waiting time.

When a root site receives a request from a neighbor, and it holds free tokens, it sends a token to the requesting neighbor via the shortest path to minimize the number of exchanged messages. If this site does not hold free tokens, but the number of tokens in the tree is sufficient to satisfy all requests after a finite amount of time, then the request will be added to the queue. If not, the site will simply propagate the request to its right neighbor.

Procedure 5: Use the shortest path(x: node)

```
1:  Begin
2:      Distance:= | i - x |
3:      If Distance k / 2 then
4:          Send Help (Token, j) to Right_Neighbor;
5:      Else
```

```
6:              Send Help (Token, j) to Left_Neighbor;
7:          End If ;
8:      End;
```

Procedure 6: Receipt of Help (Token, *j*)

```
1:      Begin
2:          If I ≠ j then
3:              Send Help (Token, j) to the appropriate neighbor
4:          Else
5:              Present_Tokens:= Present_Tokens +1;
6:                  If Length ≠ 0 then
7:                      Token Received ( );
8:                  Else
9:                      Free_Tokens:= Free_Tokens +1;
10:                 End If;
11:         End If;
12:     End;
```

After sending a token to a neighbor, the token may pass through several sites before reaching its destination; these sites may also be requesting sites, for this purpose it must be ensured that each token must be used by its requester. The Help message designates the identity of the requester, and the site receiving this message will compare its identity with the identity of the requester in the message, in case the identities are different, the site must propagate the token to the appropriate neighbor respecting the direction of sending.

Procedure 7: Receipt of Request (Token, *j*)

```
1:      Begin
2:          If Free_Tokens_ libres 0 then
3:              Use the shortest path (j)
4:              Free_Tokens:= Free_Tokens -1
```

```
5:                  Present_Tokens:= Present_Tokens -1
6:          Else
7:                  If Present_Tokens > Length then
8:                          Requester:= Requester + {j}
9:                          Length:= Length+1
10:                 Else
11:                         Send Request (Token, j) to Right_Neighbor
12:                 End If;
13:         End If;
14:   End;
```

Procedure 8: Receipt of Agreement (Token)

```
1:    Begin
2:          Status:= In SC
3:          Enter Critical Section
4:    End;
```

Procedure 9: Release the CS

```
1:    Start
2:          Status= Out;
3:          Send Liberation (Token) to Root;
4:    End;
```

Procedure 10: Receipt of Release (Token)

```
1:    Start
2:          If Lengthi ≠ 0 then
3:                  Token Received ( )
4:          Else
```

```
5:                  Free_Tokens:= Free_Tokens +1
6:          End If;
7:    End;
```

Procedure 11: Token Received ( )

```
1:    Begin
2:          Q:= the head of the queue ;
3:          Delete the head of the queue ;
4:          Lengthi:= Lengthi -1
5:          If Q is a root site then
6:                Use the shortest path (Q) ;
7:                Present_Tokens:= Present_Tokens -1;
8:          End If;
9:          If Q is a site in the tree then
10:               Send Agreement (Token) to Q
11:         End If;
12:         If Q= i then
13:               Status:= In CS
14:               Enter Critical Section
15:         End If ;
16:   End;
```

When a root receives a token from a neighbor or a site in the tree, it must update its local variables. If the queue of requesting sites is not empty, then the oldest request in that queue must be satisfied using the Received Token procedure. We distinguish three cases:

1. The head of the queue is a root, so the site has to send the token to the requesting site and still has to update the two variables: Free Tokens and Present Tokens.
2. The head of the queue is a simple site in the tree, the root will send the token and update the Free Tokens variable.
3. The requesting site is the root itself, this situation leads us to describe the behavior of a root when it wants to enter CS, if it holds a free token, it goes directly to SC. if not, it must be added to the queue to satisfy its request later.

Procedure 12: Requesting CS by a root

```
1:  Begin
2:      If Free_Tokens> 0 then
3:          Free_tokens:= Free_Tokens -1;
4:          Status:= In SC ;
5:          Enter Critical Section;
6:      Else
7:          If Present_Tokens< Lengthi then
8:              Send Request (Token, i) to Right Neighbour
9:          End If;
10:         Requester:= Requester + {i}
11:         Lengthi:= Lengthi+1
12:     End If ;
13: End;
```

Procedure 13: Releasing CS by a root

```
1:  Begin
2:      Status:= Out ;
3:      If Length> 0 then
4:          Q:= the head of the queue;
5:          Delete the head of the queue;
6:          Lengthi := Lengthi -1 ;
7:          If Q is a root site then
8:              Use the shortest path (Q) ;
9:              Present_Tokens:= Present_Tokens -1 ;
10:         Else
11:             If Q is a regular site in the tree then
12:                 Send Agreement (Token) to Q
13:             End If ;
14:     Else
15:         Free_Tokens:= Free_Tokens +1
```

16: End If;

17: End;

5. Correctness proof

5.1. The K mutual exclusion

To prove that this algorithm ensures KME, we must ensure that at most ***K*** sites execute the CS simultaneously: in algorithms using tokens, only the possession of a token allows a site to execute the CS since we use ***K*** tokens in the algorithm, then at most ***K*** sites can enter the SC. Therefore, K-EM is ensured.

5.2. Absence of deadlock

We will show the absence of interlocking by the absurd. It is assumed that the system is in a deadlock situation; therefore there is a cycle of sites *x1*, *x2*, ..., *xn* such that *x1* waits for a token of *x2*, *x2* waits for a token of *x3*, ..., and *xn* waits for a token of *x1*. In our algorithm, only roots can request tokens via the ring, a root requests a token if it cannot satisfy its local requests, before satisfying its requests, this root is not allowed to store new requests if this site receives a request, it will propagate it to a neighbor, the cycle given above can never appear in our algorithm. Therefore, the algorithm is deadlock-free.

5.3. Absence of starvation

A requesting site ***i*** will be added to the queue, all requests that arrive after this request will be added to the end of the queue. Starvation will occur if these requests get a token before site ***i,*** i.e. an unfair decision is made by the root.

In our algorithm, the root adds requests to the queue in order of arrival; the head of the queue is always the oldest request. If a free token is available, the root will send it to the head of the queue, the other requests must wait for the release of this token or the availability of another one. In our algorithm sites do not suffer from starvation.

6. Message Complexity of the algorithm

The message complexity of an algorithm is the number of required messages for each entry to the CS.

We can distinguish two situations depending on the nature of the requesting site. If the requesting site is a root and it holds a free token, it passes directly to the CS without any message exchange, otherwise, it will wait for the release of a token in its tree if it exists, or in the worst case will ask its neighbors for a token, the number of messages needed in this case is at most *K* messages. For a regular site in a tree, SC entry requires a minimum of two messages: a request message to its root, and the response message if the root holds a free token, otherwise, the root must ask for an additional token where *K* messages are needed in the worst case, so a maximum of *k+2* messages will be exchanged to respond to a site's request.

In conclusion, our algorithm requires between 0 and k+2 messages per SC input. We were able to express the complexity as a function of K and not N, which represents an interesting improvement and a very acceptable complexity.

7. The fault tolerance mechanism

Despite its interesting performance, our proposed algorithm (that we call NFT algorithm) was vulnerable to nodes failure and tokens loss. Therefore, we present a fault-tolerant version of the algorithm aiming to solve node failure, especially the failure of nodes holding tokens that causes the loss of the token.

The fault cases that may cause serious problems in our algorithm are as follows.

- The failure of a root: The local variables held in the root such as the requesting sites queue, the number of free tokens, and used tokens in the group are necessary for the proper functioning of the algorithm. If a root fails, all its variables will be lost, hence, tokens will be lost and the requesting sites can never access their CS.

- The failure of a site holding a token: In this case, the token is lost, so the number of tokens in the system decreases and will not reflect the correct number of resources, and as a result, there will be lesser sites in CS than the available resources in the system. This case affects the proper functioning of the algorithm.

- The failure of a requesting site: The identity of a requesting site is stored in the requesting sites queue. When a free token is available, it will be sent to the first site in the queue, and if this site is already crashed, then the token will be lost, and consequently, we face the same problem as the failure of a site holding a token.

The token loss is the common feature and the main disadvantage of the 2nd and the 3rd case. So to ensure the well-functioning of the system we must use complex algorithms of election to designate a site that will generate lost tokens. We aim in our algorithm to avoid the use of such algorithms.

7.1. Basic idea of fault tolerance mechanism

The basic idea of the fault tolerance mechanism used in our fault-tolerant algorithm (that we call FT Algorithm), is based on the duplication principle using the supervisor site which is a particular site designated to play a specific role whenever a failure occurs. This principle is used in many algorithms in the literature [Bou.92], [Yag.97b], and the results are very interesting.

Since roots are very important to ensure the well-functioning of the algorithm, we have appointed a supervisor to each root which is the site with the smallest identity in the group. Each supervisor will keep a copy of all variables of its root and when the root's variables are updated, the supervisor's variables must be updated as well.

In the case of a root failure, its supervisor would take its place, so the variables would not be lost and the system continues to function properly.

Depending on the nature of the site, we can distinguish four cases of failure, in which every case the algorithm must apply specific actions.

- **The failure of a requesting site**: The root removes the identity of the requesting site from the queue and informs the supervisor.
- **The failure of a site holding a token**: The root generates a new token, updates its local variables, and informs the supervisor.
- **The failure of a supervisor**: The root must choose a new supervisor; this new supervisor must be informed by all the values of the root's local variables.
- **The failure of a root**: In this case, the supervisor must act as a root, first it must inform the neighbors of the old root and the members of the group of its identity by sending a particular message, second it chooses a new supervisor, and then updates its local variables and informs the new supervisor.

In the fault-tolerant algorithm, we assume that:

- Message delays are bounded.
- Communication links are reliable and do not fail.

- A site's failure can be detected by the other nodes in the system.
- Failed sites may eventually recover.
- A root and its supervisor do not fail at the same time.

7.2. Data design

The likelihood failure of some roots or supervisors may set any site belongs to the group as a candidate to play the role of a supervisor or a root, and for this reason, the site needs the same variables of the lost root, but the difference lies in the initialization. The variables used at each site will be the same variables of the root that have been mentioned in the NFT algorithm. In addition to these variables, another two more variables are required:

- **Supervisor**: for a root, this variable contains the identity of the supervisor, for the other sites, the value is nil.
- **IN_CS**: a list containing the identities of sites holding tokens in the same group (the sites that are executing their CS). The usefulness of this variable appears in the cases of failure when a site in the list fails it is detected by the root and a new token is generated, and the identity of the failed site gets removed from the list.

7.3. Messages used by the algorithm

To update the supervisor's local variables, the root must send the new value of each variable whenever it changes, so we can use a single message containing all the variables: the left neighbor, the right neighbor, the number of free tokens, the present tokens in the group, the requesting sites queue, and the sites' list in CS.

Two problems may appear with the use of such a message. First, the size of the message could be very large, especially when there is a large number of requesting sites, so the exchange of this message would be difficult. Second, the frequency change of the different values in the message is not the same, for example, the requesting queue must be updated with every new request and the supervisor must be informed, however, the value of the variable left neighbor, for example, could remain unchanged, thus it would be unnecessary to send it each time.

For this reason, we will use a distinct message for each variable.

Update_Left_neighbor (x): x is the identity of the left neighbor.

Update_right_neighbor (x): x is the identity of the right neighboring

Update_free_tokens (x): replaces the old value of free token by x

Update_present_tokens (x): replaces the old value of present tokens by x

Update_requesting (Q): replaces the old requesting queue by Q

Update_In-CS (Q): replaces the old list by Q.

In addition to those messages, another message will be used when a supervisor takes the place of its root:

- **New_Root (i, j)**: sent by the supervisor **j** in the case of the failure of the root **i**. By this message, the members of a group and the neighbors of the old root are informed that the old root **i** has failed, and a new root **j** takes its place.

7.4. Correctness proof

Theorem 1. The algorithm guarantees K Mutual Exclusion.

Proof: To ensure KME we must ensure that at any given time K sites at most execute simultaneously the CS. Since our algorithm is a token-based, thus only the possession of a token allows a site to access the CS, and because we use K tokens in the algorithm the KME is granted. However, a problem may arise if a token holding site fails, the token will be lost consequently, and we end with at most K-1 sites in CS. To avoid this problem, in our algorithm when a token is lost the leader will generate a new one and serves the next request, therefore, the algorithm ensures the KME.

Theorem 2. The algorithm is starvation free.

Proof: Starvation may occur when a requesting site cannot access the CS in the presence of free resources, while other sites would do. This situation may happen with the failure of a leader, in such a case all the requesting sites belonging to the failed leader group will wait indefinitely because of the loss of their queue, even in the existence of free resources in the system. The fault tolerance mechanism used in our algorithm ensures that this situation would never take place, because the supervisor keeps a copy of the queue, and whenever the leader fails, the supervisor takes its place and serves all the requests in a finite time.

Theorem 3. The algorithm is deadlock-free.

Proof: The deadlock appears if there is a circular waiting chain between sites, which may be caused by the failure of a token holding site. In our algorithm, the leader generates a token

whenever it detects the failure of a token holding site in the belonged group, so the next request in the queue will be served, and the circular waiting chain between sites may never occur. Thus, the algorithm is deadlock-free.

7.5. Performance of FT algorithm

The fault tolerance mechanism of our algorithm overcomes the simultaneous failure of several sites. It can tolerate:

- The fail of K sites holding the tokens simultaneously, in this case, the leaders can detect the failure of these sites, so each leader will generate new tokens according to the number of failing sites in its group, the identity of the failing sites will be removed from the queue, and the supervisors will be informed.
- The fail of K supervisors at the same time, in this case, each leader will choose a new supervisor among the sites in its group, the new supervisors will receive the values of the local variables.
- The fail of K/2 non-adjacent leaders at the same time, the supervisors of the failing leaders will take place; the failing leaders must not be adjacent to avoid the conflicts that may occur when the supervisors take place of the failing leaders.

In all these cases, our algorithm ensures that the system does not cease to function smoothly. Moreover, the lost tokens will be generated by the leaders, avoiding the use of complex election algorithms to identify the site that will generate the lost tokens, our principle makes easier this task and minimizes the number of exchanged messages while ensuring KME and fault tolerance.

8. Simulation results

The performance of the NFT algorithm was shown in [T.al.08], the behavior of our algorithm compared to Raymond's algorithm [Ray89], and Srimani and reddy algorithm [Sri.92] is very interesting with a better message complexity.

In this section, we present the simulation results of the proposed FT algorithm. We compared its behavior with that of the NFT algorithm by causing intentionally some faults in the system. To create different scenarios, we varied one of the metrics each time: the number of resources, and the number of requesting sites. The simulation compares the message complexity and the waiting delay between the FT algorithm and the NFT algorithm.

In Figure 7 we notice that the number of messages for each entrance to CS decreases with the increase of the number of resources for both algorithms, however, the FT algorithm requires more messages comparing with the NFT algorithm due to the exchange of messages between new leaders and other sites when a fault occurs.

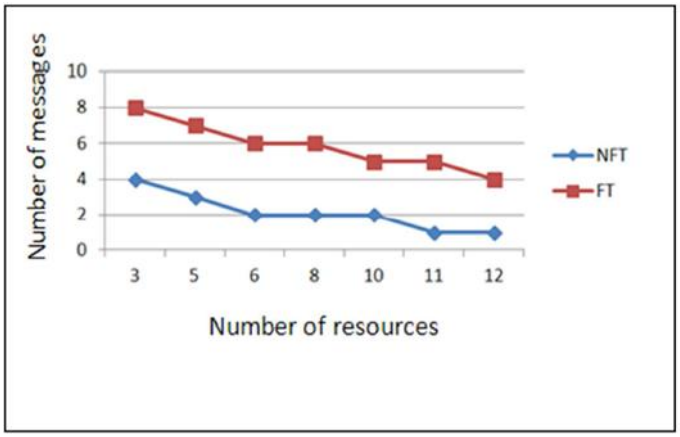


Figure 7. Number of messages versus number of resources

In Figure 8 the waiting time curve has the same direction of change with that of Figure 1 because the increase of the number of resources minimizes the load on leaders and thus requests will be satisfied in a shorter time. We notice that the difference between the NFT curve and the FT curve is not considered because the extra exchanged messages haven't an influence on the waiting time.

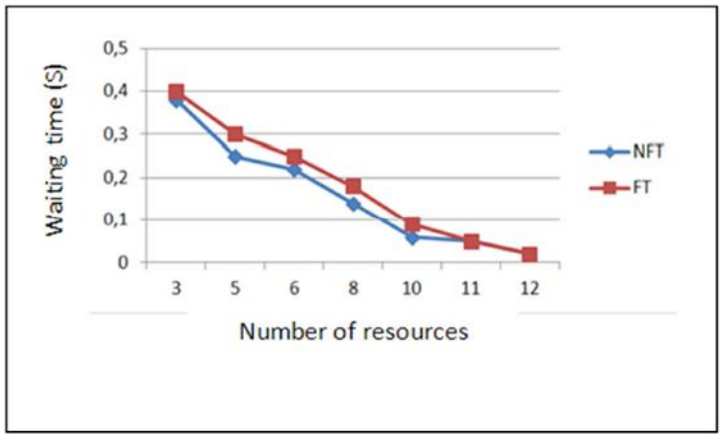


Figure 8. Waiting time versus number of resources

In Figure 9 we notice a slight increase in the number of messages with increasing of the requesting nodes, for the same reason of the precedent curves, the FT algorithm requires a little more exchanged messages. Curves in Figure 10 are practically identical.

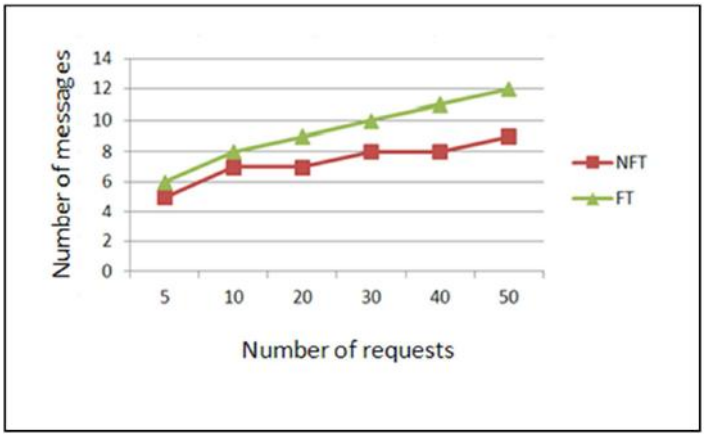


Figure 9. Number of messages against number of requests

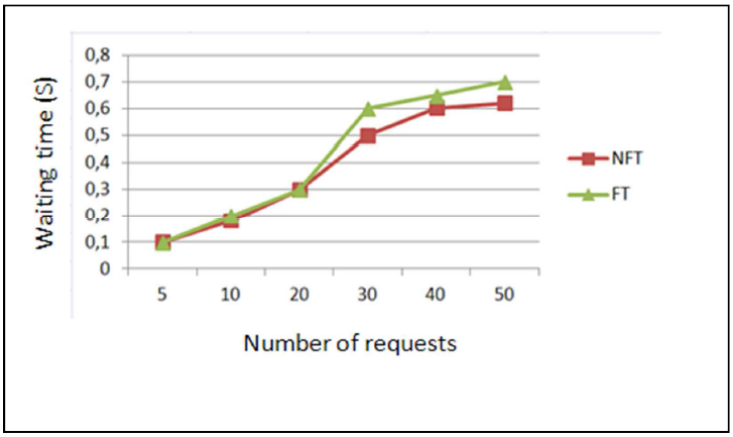


Figure 10. Waiting time against number of requests

In all figures, we notice that the FT algorithm has the same behavior with the NFT algorithm despite a light inevitable difference in the messages number because of the necessary exchange of information between the new leaders and the other sites in case of a fault. we conclude that the fault tolerance mechanism that we use in our algorithm keeps the same behavior and performance of the initial algorithm, the waiting time remains practically the same, and the difference in the number of exchanged messages does not influence the interesting performance of the algorithm.

9. Conclusion

In this chapter, we presented an efficient fault-tolerant K mutual exclusion algorithm in distributed systems, the algorithm ensures the K mutual exclusion by using K tokens, and each access to the critical section needs at most K+2 messages.

In our proposal, we used the supervisor mechanism to make the algorithm fault-tolerant, this mechanism resists several nodes crashes and the loss of tokens as well; therefore, no complex election algorithms are needed.

The main contribution of this algorithm is the development of a fault-tolerant mechanism that ensures the KME and overcomes the likelihood of token loss. Simulation results show that message complexity between FT and NFT algorithms has no significant difference. However, our study was only limited to token loss, thus the algorithm needs to be more improved to deal with other fault types such as messages loss and communication link failure. These two problems can be considered in future improvements.

In the next chapter, we will deal with the same problem in mobile ad hoc networks, and a new fault-tolerant algorithm for such an environment will be presented.

Chapter 5:

A fault-tolerant K-ME algorithm in mobile Ad hoc networks

1. Introduction

Nowadays, the development of wireless technologies offers new perspectives in the field of computing. Wireless networks allow users to gain access to information independently of time and place. Due to their many advantages, they gained major interest and increasing popularity in both academic and industrial domains [Rap.96],[Pra. 17].

An Ad hoc network is a set of mobile hosts interconnected by a wireless medium, forming a temporary network without any communications infrastructure support. The mobile ad hoc networks concept assumes that all (or the majority) of the components in the environment are mobile. Therefore, contrary to infrastructure-based networks, no centralized administration is available; instead, mobile hosts themselves form the infrastructure of the network. Also, no assumption or limitation is made regarding the size of the mobile ad hoc network, meaning that the network can potentially contain hundreds of mobile units [Nas.04]

The absence of a centralized administration in ad hoc networks, and the possibility of sharing resources between the various sites of the network, can jointly lead to incoherences due to simultaneous access to the same resource from several locations, and so it becomes of utmost importance to guarantee the exclusive access to this resource. This notion gave birth to the mutual exclusion problem (MUTEX) in mobile ad hoc networks [Nas.04]. This problem can be generalized to the K-mutual exclusion problem (K-MUTEX) when the system is endowed with several copies of the same resource [Bul.95].

These two problems were adequately addressed in conventional distributed systems, and the proposed algorithms can be classified into two main categories:

1. permission-based algorithms
2. token-based algorithms

Many algorithms of these two categories have been proposed [Ben.04], most of these solutions ignore the possibility of site failures [Wal.97], [Wal.98], [Wal.01]. However, the assumption that all nodes are safe from any kind of fault can lead to unexpected and unwanted situations.

The purpose of the proposed algorithms is to ensure the K-MUTEX through the exchange of a reduced number of messages to avoid overloading the network and to reduce energy consumption. Unfortunately, fault tolerance is rarely considered.

In this chapter, we propose a new algorithm to solve the K-MUTEX problem in mobile ad hoc networks that supports fault tolerance [T.al.19]. Our solution takes advantage of the

AODV routing protocol [Cha.04] to achieve a reduced exchange of messages between the various mobile hosts whenever access to a critical resource takes place. Moreover, we extend AODV control messages to carry both critical resource availability and fault tolerance information.

After ensuring the distribution of token availability data on the fly (together with control messages), our proposal offers a double-layer recovery strategy to face any kind of nodes failure. Thus, our proposal (which we named NFK), is an AODV-based, fully distributed, timely, and fault-tolerant K-ME solution for mobile ad hoc networks (MANETs).

2. Basic idea

In MANETs, all the applications that require communication between two hosts require at first the establishment of a link between these two hosts. This is ensured by the routing protocol of the network. The establishment of links is insured by the routing protocol at the network layer, while the various applications, including those of the K-MUTEX, run at the application layer.

For token-based K-MUTEX algorithms, the requester sends its demand to a single particular site, or it sends its demand towards a set of sites and waits for the answer. In both cases, a high number of message exchanges between sites will take place, leading to network overload. Moreover, to ensure this communication, a valid path must be already established by the routing protocol, which also requires message exchanges.

We notice that the use of the shared resource requires an exchange of messages in the lower layer, and another exchange of messages in the upper layer as well. Thus, it quickly becomes obvious that network overload may take place.

Our approach consists of reducing the number of exchanged messages for every CS entry, and of exploiting the inevitable messages of the routing protocol to carry additional information that ensures token exchanges between the various sites.

To solve the problem of the K-MUTEX, K tokens are used to satisfy the requests. These tokens are initially held by K sites and can be exchanged between the various sites. Thus, at any given time, every site can be in one of the following statuses:

1. the site holds a free token
2. the site uses a token
3. the site does not hold a token.

Our idea consists of adding two fields in the routing table of each site to indicate the presence and the usage of tokens. The first field is the Present_token field; it indicates that the site holds a free token. The second field is the Used_token field, which indicates that the site uses a token.

We rely on the AODV routing protocol because it is a reactive algorithm that allows establishing a link between two network nodes by using the shortest path. To achieve this it creates routing tables on demand, assuring that the information is always relevant.

The site holds a token, meaning that this information will be available in its routing table, and by dint of the routing protocol, this information will be available in the routing tables of the other sites.

When a site wishes to enter the CS, it updates its routing table, and it chooses the closest site among the sites that hold tokens. This information is available in the routing table, and thus a request will be directly sent towards this site without the need to perform a token search. The site that receives the request answers favorably to this demand by returning the token. Upon receiving the token, the requesting site allows it to access the CS. Notice that all necessary updates are handled automatically by the routing protocol.

It is worth pointing out that the use of this method avoids additional message exchanges because the information concerning tokens and their location is in general available for all the sites in their routing tables.

3. Proposal details

The system consists of ***N*** sites numbered from 1 to N, and of ***K*** resources. To ensure the K-MUTEX, we use ***K*** tokens that are initially held by the sites in a random manner.

3.1 Data design

Every node ***i*** in the system maintains the following local variables:

- *Status*: a variable that indicates the status of the site.
- *Requesting*: a Boolean variable indicating if the site asked a critical resource or not, initialized at false.
- *Token*: a Boolean variable indicating if the site possesses a token or not; it is initialized at True for the sites holding tokens, and at false for the other sites.

- *Next*: a waiting queue that contains the identities of the requesting sites. This queue is sent with the token to a requesting site to satisfy the other sites.
- *AODV (X, Y)*: indicates both the fields which can be manipulated by our algorithm: X indicates the free tokens, and Y indicates the used tokens.

In our algorithm, we use also the following messages:

- *Request (Token, i):* sent by the site i towards the site that holds a token.
- *Agreement (Token, Queue):* sent by the site which holds the token to a requesting site to allow it to use the CS.
- *Reject*: sent by a former holder of the token to a requesting site to inform it that it does not hold the token anymore.

3.2 Proposal functionalities

In this section, we present the code of the algorithm with the explanation of each procedure.

In Procedure 1, the requesting site sends its demand to the closest holder of a free token, and waits for its reception to enter the CS.

Procedure 1 Requesting the CS

```
1:  Requesting:= True;
2:  if (AODVi_[i]_free token = 0) then
3:      Holder := Find the adequate holder (AODVi);
4:      Send Request (Token, i) to Holder;
5:  end if
6:  Wait for (Token = True);
7:  Modification in the routing table AODVi (0,1);
8:  < Enter the CS >
```

In addition, when a site receives an agreement message, it sets its local variable at true, and updates the waiting queue. Procedure 2 summarizes this process.

Procedure 2 Reception of agreement (Token, Queue)

```
1: Token := True;
2: Next := Queue;
```

Using Procedure 3, if a site receives a request message, it is going to send the token towards the requesting site if it does not use the resource. Otherwise, it puts the requesting site

on hold. In case the site does not hold a token anymore, it is going to send a reject message to the requesting site.

Procedure 3 Reception of request (Token, j)

```
1:  if (AODVi_[i]_free_token = 1) then
2:      Token := False;
3:      Send Agreement (Token, Queue) to j;
4:      Modification in the routing table AODVi (0,0);
5:  else
6:      if (AODVi_[i]_present_token = 1) then
7:          Next:= Next+ {j};
8:          Queue:= Next;
9:      else
10:         Send Reject () to j;
11:     end if
12: end if
```

Following Procedure 4 when a requesting site receives a reject message, it is going to look for another adequate site holding a free token by consulting its routing table.

Procedure 4 Reception of Reject ()

```
1:  Holder:= Find the adequate holder (AODVi);
2:  Send Request (Token, i) to Holder
```

When a site releases the critical section, it sends the token with the waiting queue to the first requesting site in that queue. If the waiting queue is empty, the token remains held by this site. After that, an update of the routing table must be done as shown in Procedure 5.

Procedure 5 Release the CS

```
1:  Requesting := False;
2:  if (Not_empty (Next)) then
3:      j := head(Next);
4:      Next := Next – {j};
```

```
5:              Queue :=  Next;
6:              Send Agreement (Token, Queue) to j;
7:              Token:=   False;
8:      else
9:              Modification in the routing table of AODVi (1,1);
10:     end if
```

In Procedure 6, the modification in the routing table concerns two variables: the existence of a free token, and the existence of a present token, according to the values of X and Y.

Procedure 6 Modification in the routing table AODVi (X,Y)

```
1:      AODVi_[i]_Free_token  := X;
2:      AODVi_[i]_present_token :=  Y;
```

Procedure 7 presents the judicious manner that we use to find the closest token holder for the requesting site.

Procedure 7: Find the adequate token holder(AODVi)

```
1:      X :=  Y;
2:      for Q := 1 to N do
3:              if (AODVi_[Q]_free_token = 1) then
4:                      Min :=  AODVi_[Q]_Distance;
5:                      if Min < X then
6:                              Holder:=  Q;
7:                              X :=  Min;
8:                      end if
9:              end if
10:     end for
11:     if (X = Y) then
12:             for Q := 1 to N do
13:                     if (AODVi_[Q]_present_token = 1) then
14:                             Min:=  AODVi_[Q]_Distance;
15:                             if Min < X then
```

```
16:                                Holder := Q;
17:                                X := Min;
18:                            end if
19:                        end if
20:                    end for
21:                end if
22:                Return (Holder);
```

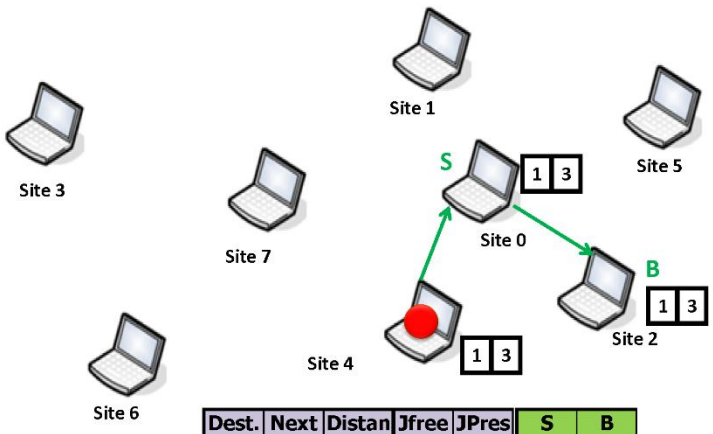
5. K-MUTEX with fault tolerance

In Ad hoc networks, any site may crash at any time for several reasons. This site may contain important information for the well-functioning of the system. In this case, this fault will cause degradation to the functioning of the whole system. Thus, a fault tolerance mechanism is needed to ensure that the system operates properly, even in the presence of faults. This is achieved by choosing backup sites that may take action when some sites crash. In our algorithm, the major problem is the loss of the token or the waiting queue. This loss is caused by the crash of a site holding these two information elements, and so we must ensure that these two data structures are not lost, and if they are lost, the system should be able to regenerate them.

5.1. Basic idea

In our proposal, a double-layer fault-tolerance mechanism is used to ensure that the system continues to function normally when the sites holding important information crash. Our idea consists in associating with every site holding a token another site called the safe site. This safe site keeps a copy of the waiting queue, and it can even generate a new token when the site holding the token crashes. This duplication allows the system to continue to work properly even after a loss because the critical information to guarantee its functioning is not lost thanks to the second copy held by the safe site. However, notice that the safe site is just another site, and so it may itself crash at any time. This will cause the information held in it to be lost, the reason why we add a second layer of recovery to avoid such situations. Hence, for every safe site, we choose another site called the back-to-back site. This site keeps another copy of the waiting queue, and it can act as a safe site when the former crashes. Thus, using our fault tolerance mechanism, we have the site that holds the token and the waiting queue, and for this site, a safe site is chosen, and a back-to-back site is chosen for every safe site. This way we can ensure that important information will not be lost. The choice of these sites can be based on distance. The

safe site and the back-to-back site must be the closest and most stable ones to minimize the number of hops and to avoid packets loss in the network. Besides, the selected site should not be used by another site holding token. Figure 11 illustrates the proposed message format extension to ensure the fault-tolerance.



Dest.	Next	Distan	Jfree	JPres	S	B
0	4	1	0	0	-	-
1	0	2	0	0	-	-
2	4	1	0	0	-	-
3	4	1	0	0	-	-
4	4	0	0	1	0	2
5	0	2	0	0	-	-
6	4	1	0	0	-	-
7	4	1	0	0	-	-

Figure 11. Proposed message format extension related to the fault-tolerance

5.2 Fault situations

We can find several cases of faults in our system, where the fault can occur at:

- the token holder
- a safe site
- a back-to-back site
- both a site holding a token and its safe site at the same time
- both a site holding a token and a back-to-back site at the same time
- both a safe site and its back-to-back site at the same time.

We will now discuss the actions that must be taken for every situation.

5.2.1 A site holding a token

When a failure of a site holding a token is detected by its safe site, the latter must act as a token-holding site by following these steps:

1. generate a new token
2. remove the identity of the failing site from the waiting queue
3. send the new token with a copy of the waiting queue to the first requesting site in the queue
4. propagate the information to its back-to-back site.

Figure 12 illustrates the tolerance strategy when the fault occurs at the token holder level.

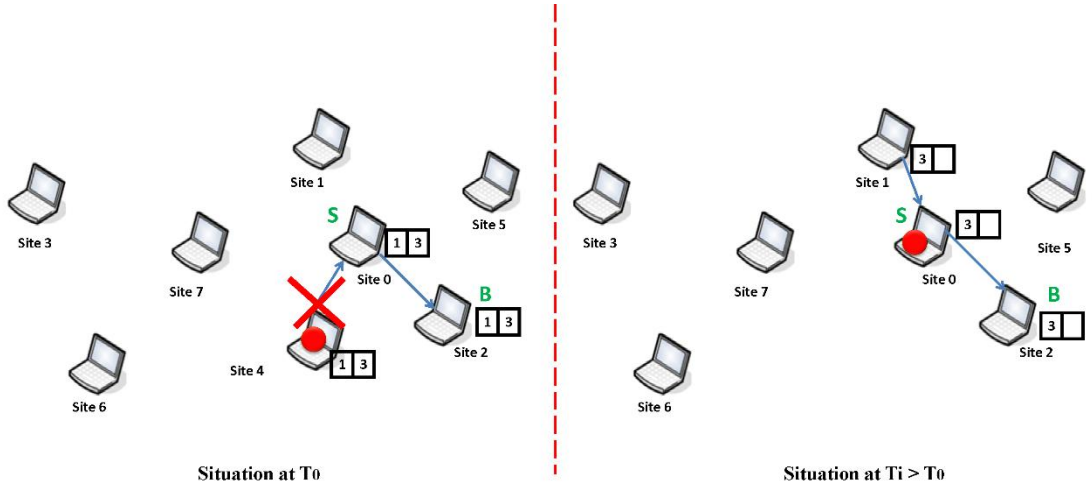


Figure 12. Tolerance strategy when the fault occurs at the token holder site

5.2.2 A safe site

In this case, the back-to-back site becomes a safe site, and it chooses a new site to become its back-to-back site. The latter will receive a copy of the waiting queue from its safe site, as illustrated in Figure 13.

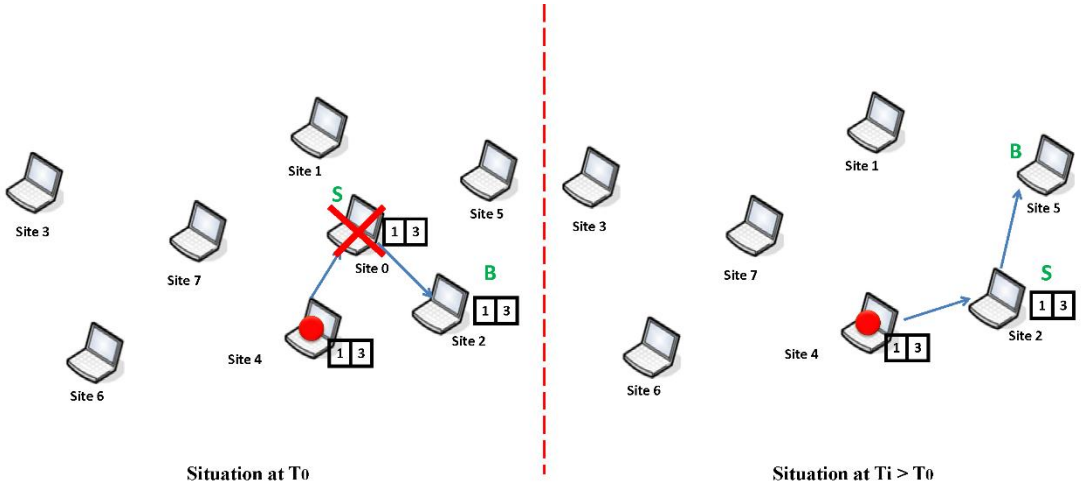


Figure 13. The tolerance strategy when the fault occurs at the safe site

5.2.3 Back-to-back site

The safe site chooses a new back-to-back site, and sends a copy of the waiting site to it, as illustrated in Figure 14.

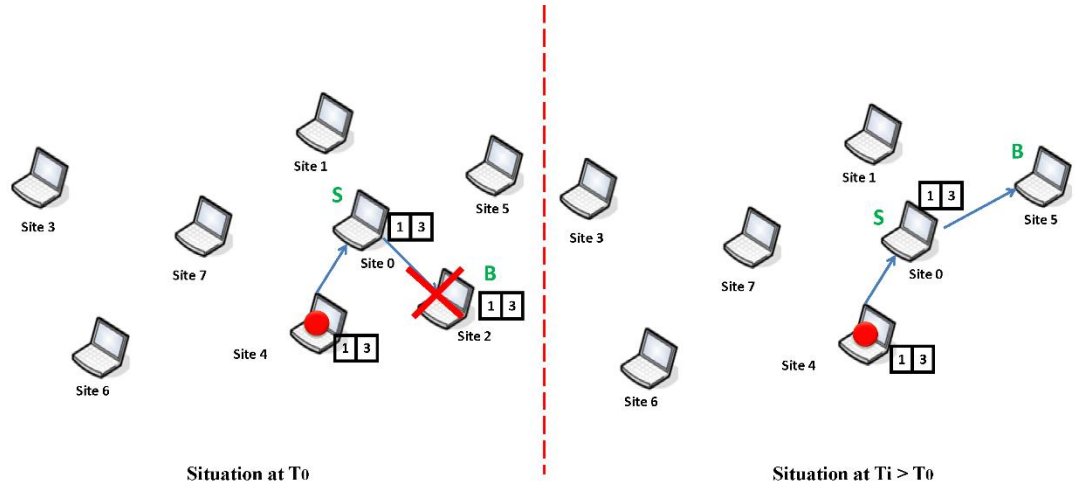


Figure 14. Tolerance strategy when the fault occurs at the back-to-back site level

5.2.4 A site holding a token and its safe site at the same time

The back-to-back site removes the identities of the failing sites from the queue if they are present in it, generating a new token, and then sending the token with a copy of the waiting queue to the first site in the queue. Figure 15 illustrates the tolerance strategy when the fault occurs at both the token holder and its safe site at the same time.

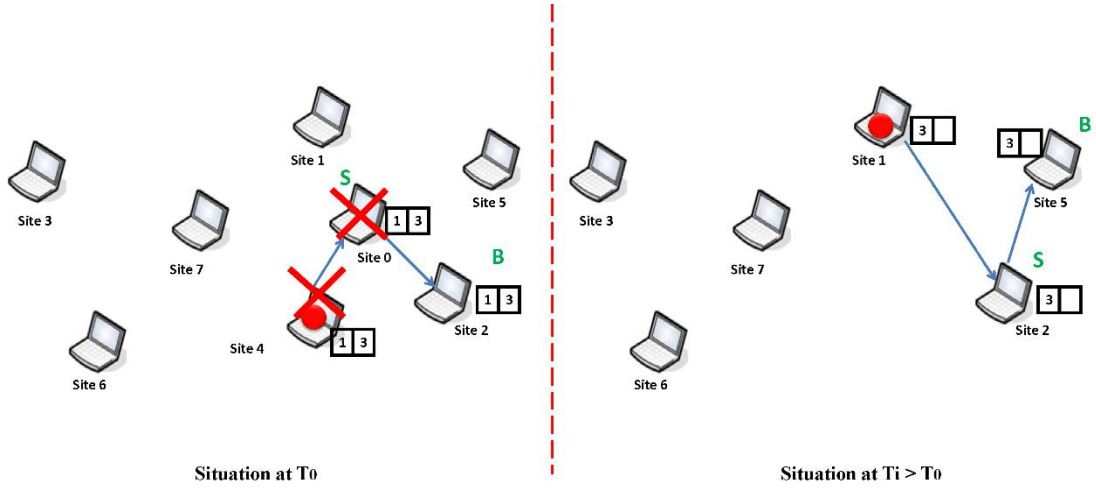


Figure 15. Tolerance strategy when the fault occurs at both the token holder and its safe site at the same time

5.2.5 Site holding a token and a back-to-back site at the same time

The safe site removes the identities of the failing sites from the queue if they exist in it, generate a new token, and sending the token with a copy of the waiting queue to the first site in the queue. Afterward, it selects a new back-to-back site.

Figure 16 illustrates the tolerance strategy when the fault occurs at both a site holding a token and a back-to-back site at the same time.

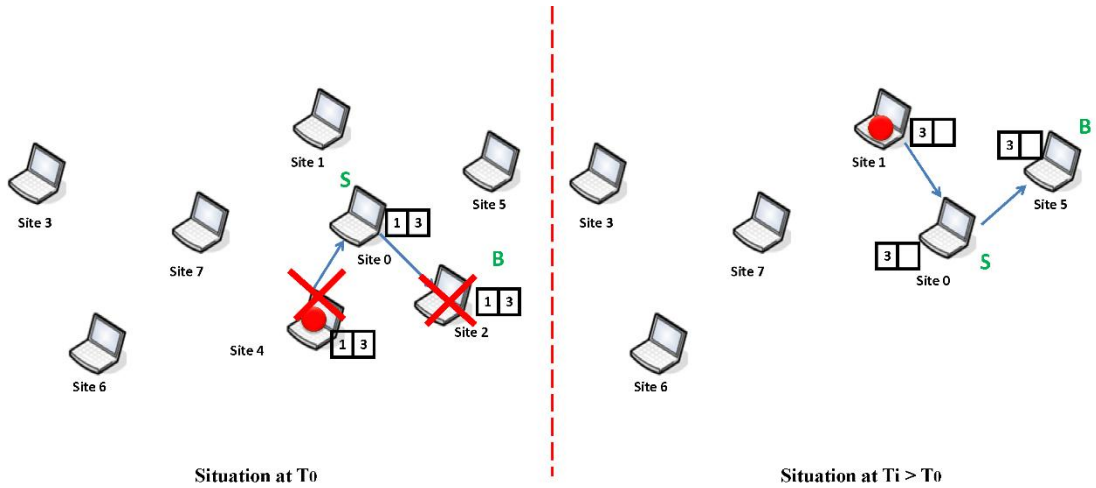


Figure 16. The tolerance strategy when the fault occurs at a site holding a token and a back-to-back site at the same time

5.2.6 A safe site and its back-to-back at the same time

The site holding the token chooses a new safe site and sends a copy of the waiting queue to it. The latter will choose a site as its back-to-back site and sends a copy of the waiting queue.

Figure 17 illustrates the tolerance strategy when the fault occurs at both a safe site and its back-to-back site at the same time.

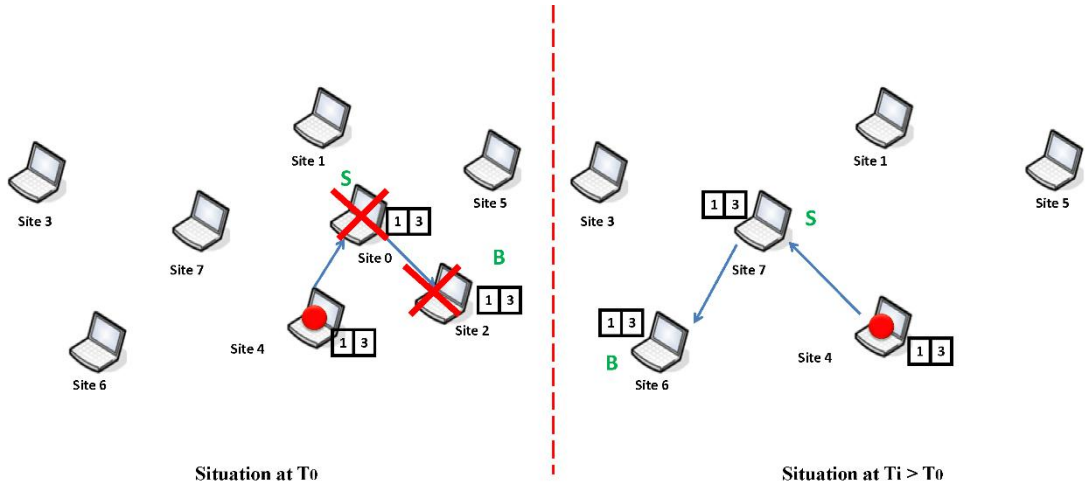


Figure 17. The tolerance strategy when the fault occurs at a safe site and its back-to-back site at the same time

6. Performance evaluation

To evaluate our proposal we relied on the NS-2 simulator using the IEEE 802.11 standard for communications. Simulations run during 300 seconds, during which mobile nodes move with a variable speed ranging from 0 to 8 m/s, within a 1 km2 area using the Random Way Point mobility model. Furthermore, randomly chosen ***K*** nodes hold tokens.

We evaluated the performance of our solution on different scenarios by varying: number of requests, network density, communications range, and number of faults. Also, for every scenario, we studied the amount of exchanged messages together with the generated delay. We also compared the waiting time and number exchanged messages of our proposal NFK to those obtained by CKM [Hag.11], we chose to compare with CKM mainly because it was evaluated within the same environment like ours and using the same routing strategy.

Figure 18 presents the amount of exchanged messages concerning the number of requests both with and without fault-tolerance mechanisms. The average number of messages is much higher for the algorithm with fault tolerance because of the different updates required to

implement the required tolerance. However, this double-layer system achieved a similar delay despite the presence of faults, as illustrated in Figure 19.

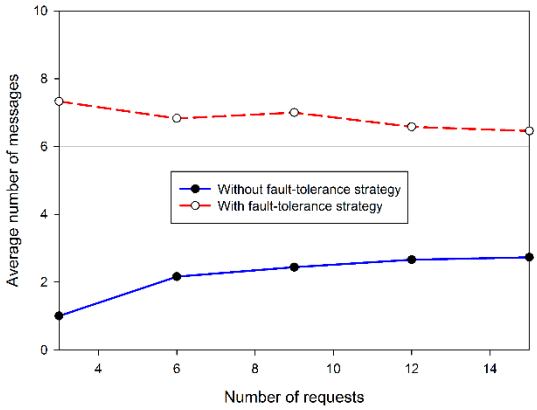


Figure 18. Amount of exchanged messages with respect to the number of requests

Besides; in the first part, Figure 19 also depicts that the existence of failure had a positive effect on certain sites, since if a token-holding site fails, the token is sent to the waiting requesters and then, in the second phase, they experience higher waiting times than the algorithm without tolerance, which is quite logical considering the time taken to resume operation after failure. Furthermore, Figure 19 shows also that thanks to the efficient AODV-based on-the-fly CS information sharing, our full proposal with fault tolerance takes almost the same waiting time as CKM which does not consider any fault tolerance mechanism.

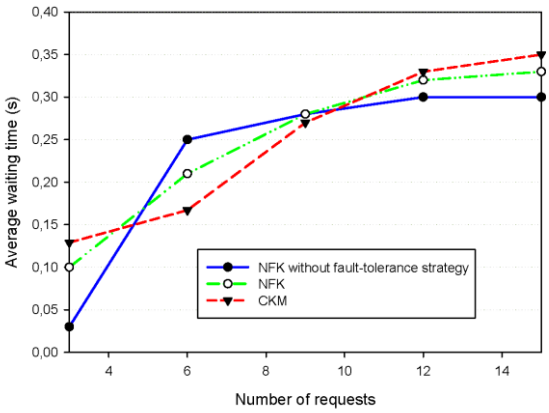


Figure 19. Average waiting time to enter the CS of NFK compared to CKM

We can see from the variation of the communications range in Figure 20 that the amount of exchanged messages logically decreases when the communications range increases since, in this situation, end-to-end communications require fewer hops. Whereas, the cluster-based approach adopted by CKM without fault tolerance requires almost the same number of messages only to ensure the K-MUTEX.

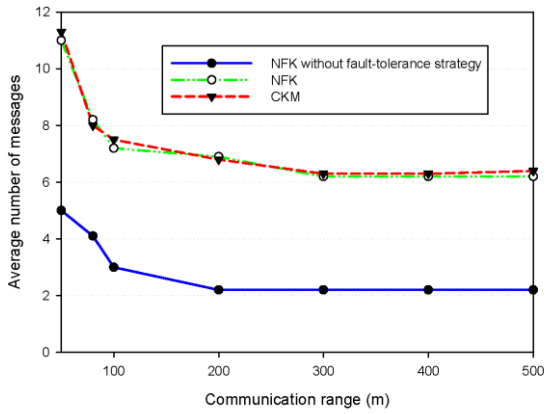


Figure 20. Amount of messages exchanged of NFK compared to CKM

Furthermore, the existence of breakdowns has not affected the average waiting delay, as illustrated in Figure 21, thanks to our efficient and fast double-layer recovery strategy.

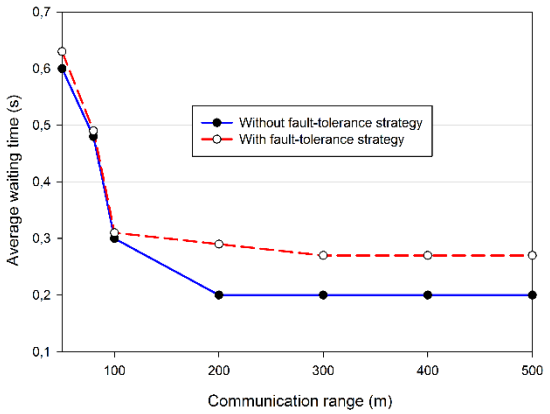


Figure 21. Average waiting time to enter the CS with respect to the communications range

Figure 22 shows an increasing behavior in terms of the number of exchanged messages concerning the number of sites because of the number of updates to each failure.

However, the waiting time for the fault tolerance algorithm also increases due to the immediate regeneration after a token holder site fails (see Figure 23). This occurs because the duration of the critical section is the most influential parameter in the waiting time.

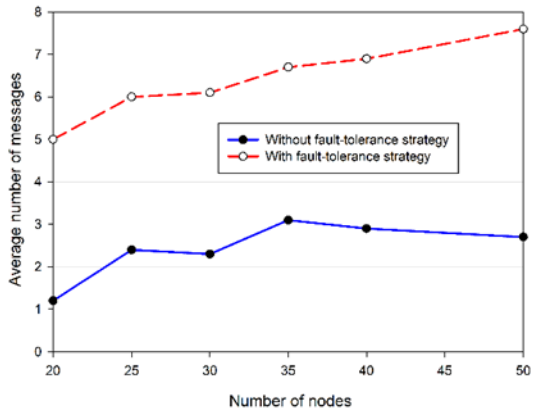


Figure 22. Amount of exchanged messages with respect to network density

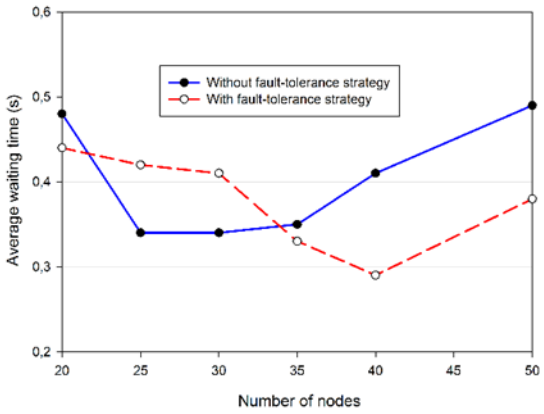


Figure 23. Average waiting time to enter the CS with respect to network density

In Figure 24, the unstable behavior of the curves has a direct relationship with the number of fails of sites acting as token holders, safe or back-to-back. Hence, the wait time is logically higher for the algorithm providing fault tolerance (see Figure 25). Also notice that, since we randomly generated the identities of the nodes where faults occur, multiple token holders can breakdown at the same time.

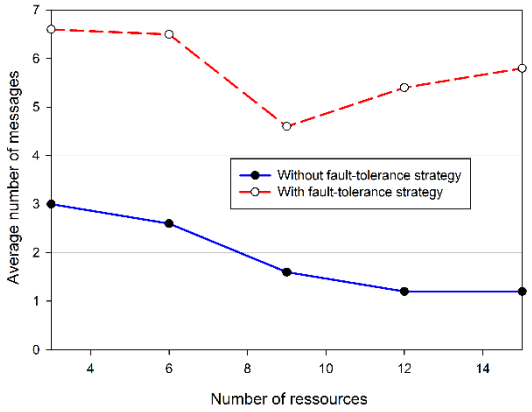


Figure 24. Amount of exchanged messages with respect to the number of resources

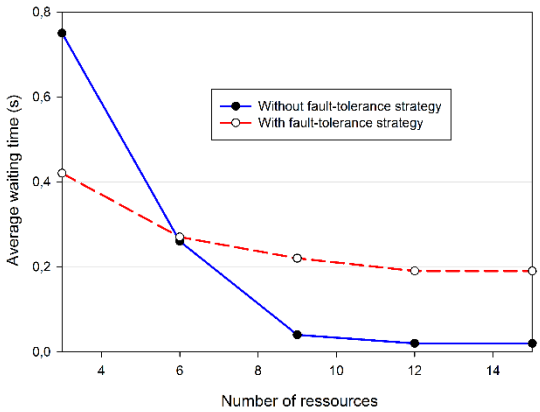


Figure 25. Average waiting time to enter the CS with respect of the number of resources

From both Figures 26 and 27, we can see that the number of messages and the waiting time increase logically when increasing the number of faults due to the different fault-tolerance updates. However, our proposal can sustain low delays (<0.4 seconds in the worst case).

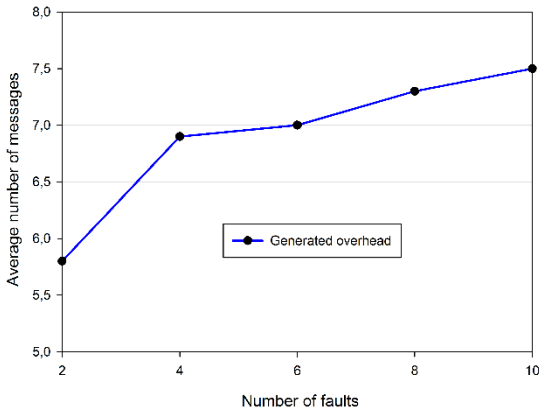


Figure 26. Amount of exchanged messages in respect of the number of faults

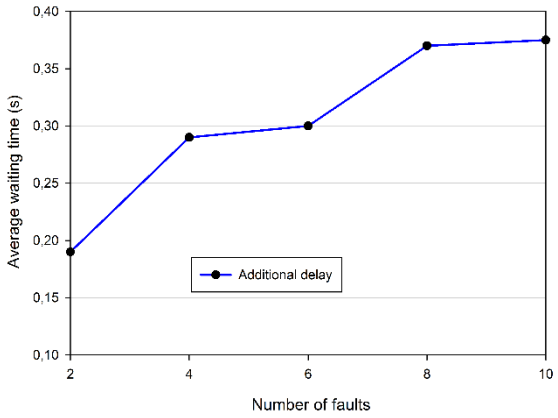


Figure 27. Average waiting time to enter the CS with respect to the number of faults

7. Conclusion

In this chapter, we presented a new K-MUTEX algorithm for mobile ad hoc networks.

We used a new method that can be considered as an innovation in this domain. In particular, our method integrates the information concerning the availability of tokens in the routing tables to minimize the number of exchanged messages.

We used a fault-tolerant mechanism based on the duplication of information to resist token losses in our system. This dual-layer mechanism overcomes token loss and permits the well-functioning of the system even when several sites fail at the same time.

Our study of fault tolerance was limited to the token loss. Other situations of failure include message losses and links failures caused by the network partitioning. These issues will be addressed in future works. Simulation results show that our algorithm guarantees the accesses to the CS while exchanging a reduced number of messages, and introducing a very short waiting time.

Conclusion and Perspectives

Distributed systems and Mobile Ad hoc networks have proved their place and importance in today's world, and the problems associated with these systems remain topical, in particular the problem of mutual exclusion and K-mutual exclusion.

In this thesis, we have presented these problems in detail; we started with the simple mutual exclusion problem, detailing the categories of proposed solutions, as well as the most referenced algorithms. We also presented the problem of K mutual exclusion, while explaining the most cited algorithms.

We also presented the concept of dependability in computer systems and explained the different methods used to ensure it, especially fault tolerance.

The study of these problems and their solutions allowed us to identify the shortcomings in the algorithms already proposed in the literature, thus, to propose new and more efficient solutions.

We first have proposed a new solution to the K mutual exclusion distributed systems; this algorithm is toke based, and combines two different logical structures to minimize the number of exchanged messages for each entry to the CS.

The complexity of our algorithm depends on K instead of n, and the number of exchanged messages per SC input varies between 0 and K+2, which represents an interesting complexity.

This algorithm has been enhanced with a fault tolerance mechanism that allows the system to continue to operate even if several sites in the system stop working for one reason or another.

We then proposed and simulated a new K-mutual exclusion algorithm in mobile Ad-hoc networks. The principle of this algorithm is new and consists of integrating additional information in the messages used by the AODV routing protocol to minimize the high message exchange for each CS entry.

The algorithm was then improved by the addition of a new fault tolerance mechanism, and the simulation gave very acceptable results.

This work has allowed us to study and understand the problem of resource sharing in distributed systems and Manets. Moreover, to have an idea about fault tolerance and the advantages it offers for distributed systems.

The interesting results obtained with our algorithms encourage us to exploit their principles in other fields, for this purpose, we will propose the following perspectives:

- Use other routing protocols and make a comparison to get the best possible method in the NFK algorithm.
- Use coteries as a method of partitioning the network and make a comparison with the method used in the FT and NFT algorithm.
- Adapt the operating principles of our algorithms in other systems such as Vanets and Fanets.

References

Agr.91 D. Agrawal, A. El Abbadi,
An efficient and fault-tolerant solution for distributed mutual exclusion,
ACM Transactions on Computer Systems, Vol. 9, No 1, February 1991, 1-20

Bal. 95 R. Baldoni, Y. Manabac, M. Raynal, S. Aoyagi,
k-Arbiter: A safe and general scheme for h-out of k-mutual exclusion problems,
INRIA, Rapport de recherche n° 2523, Avril 1995

BCi. 94 R. Baldoni, B. Ciciani,
Distributed algorithms for multiple entries to a critical section with priority,
Information Processing Letters 50, 1994, 165-172.

Ben.04 M. Benchaiba, A. Bouabdallah, N. Badache, and M. Ahmed-Nacer,
Distributed mutual exclusion algorithms in mobile ad hoc networks : an overview,
ACM SIGOPS Operating Systems Review, 38, 2004, 74–89.

Bou.92 Abdelmadjid Bouabdallah, Jean-Claude König,
An Improvement of Maekawa's Mutual Exclusion Algorithm to Make It Fault-Tolerant,
Parallel Process. Lett. 2: 1992, 283-290.

Bul. 95 S. Bulgannawar, N.H. Vaidya,
Distributed K-mutual exclusion algorithm,
Proceedings of the 15th International Conference on Distributed Computing Systems, Vancouver, Can, 1995, 153-160.

Car.83 O.S.F. Carvalho, and G.Roucairol,
On mutual exclusion in computer networks,
Comm. ACM, Vol. 26, N°. 2, 1983, 146-147.

Cha.04 I.D Chakeres, E.M Belding-Royer,
AODV routing protocol implementation design,
Proceedings: 24th International Conference on Distributed Computing Systems Workshops, 2004, 698–703.

Cou.94 G. Coulouris Dollimore,
Distributed Systems: Concepts and Design,
International Computer Science Series by Jean Kindberg Tim (1994-04-01)

CSL.91 Y.Chang, M. singhal , M. liu
A dynamic token-baseddistributed mutual exclusion algorithm.
Proceedings of the 10th IEEE international phoenix conference on computer and communications, March 1991, 240-246.

Dag.07 Dagdeviren, K. Erciyes,
A software architecture for shared resource management in mobile ad hoc networks,
SOFSEM: Theory and Practice of Computer Science, 2007,224–234.

Dji.65 E.W. Dijkstra,
Solution of a problem in concurrent programming control,
Communications of the ACM, 8, 1965, 569.

Erc.04 K. Erciyes,
Cluster based distributed mutual exclusion algorithms for mobile networks,
Euro-Par ,Parallel Processing,2004, 933–940.

Hag.11 A.T. Haghighat, M.R. Mohamadi,
Cluster-based k mutual exclusion for mobile ad hoc networks,
5th International Conference on Application of Information and Communication Technologies, (AICT), 2011.

Hel.94 Jean-Michel Hélary , Achour Mostefaoui,
A o(log 2 n) fault-tolerant distributed mutual exclusion algorithm based on open-cube structure,
14th International Conference on Distributed Computing Systems (ICDCS), 1994, 89-96.

Hos.01 Tréhel, Michel , Ahmed Housni,
The Prioritized and Distributed Synchronization in Distributed Groups,
International Conference on Computational Science, 2001.

HPR.88 Jean-Michel Hélary, Noël Plouzeau, and Michel Raynal,
A distributed algorithm for mutual exclusion in an arbitrary network,
Computer journal,31(4) : 1988, 289-295.

Jia. 03 J.R. Jiang,
A prioritized h-out of-k mutual exclusion algorithm with maximum degree of concurrency for mobile Ad hoc networks and distributed systems,
Proceedings of the 4th PDCAT, 2003, 329-334.

Jou.98 Y.J. Joung,
Asynchronous group mutual exclusion algorithm (extended abstract),
Proc. in the 17th Annual ACM symposium on Principles of Distributed Computing, 1998

Kak. 94 H Kakugawa, S Fujita, M Yamashita, T Ae,
A distributed k-mutual exclusion algorithm using k-coterie,
Information Processing Letters 49 (4), 1994, 213-218.

Lam.78 L.Lamport,
Time, clocks, and the ordering of events in a distributed system,
Comm. ACM, Vol. 21, N°. 7, July 1978, 558-565.

Lap.92 J.C. Laprie,
Dependability: Basic Concepts and Terminology,
Springer-Verlag, 265 p., ISBN 3-211-82296-8 and 0-387-82296-8, 1992

Lel.77 G. Le lann,
Distributed systems, Towards a formal approach,
IFIP Congress, Toronto, Canada, 1977, 155-160.

Lel.78 G. Le Lann,
Algorithms for distributed data-sharing systems which use tickets,
In Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, August 1978, 259-272.

Mae.85 M. Maekawa,
An √n Algorithm for Mutual Exclusion in Decentralized Systems,
ACM Trans. Computer Systems, Vol. 3, N°. 2, 1985, 145-159.

Mas.10 Masum, S.M., Akbar, M.M., Ali, A.A. and Rahman, M.A,
A consensus-based ℓ-exclusion algorithm for mobile ad hoc networks,
Ad Hoc Networks, Vol. 8, No. 1, 2010, 30–45.

Mat.88 F. Mattern,
Virtual time and global states in distributed systems,
International Conference on Parallel and Distributed Algorithms, North Holland, 1988, 215-226.

MOA.07 Moallemi, M., Moghaddam, M.H.Y. and Naghibzadeh, M.
A fault-tolerant mutual exclusion resource reservation protocol for clustered mobile ad hoc networks,
Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, andParallel/Distributed Computing, 2007, 528–533,

Nai. 93 M.Naimi,
Distributed algorithm for *K*-entries to a critical section based on the directed graphs,
ACM, Op. Systems Review, Oct 1993

Nas.04 Nasipuri, A.
Mobile ad hoc networks',
Handbook of RF and Wireless Technologies, 59–100, Elsevier, USA.

Nel. 91 Mitchell L. Neilsen and Masaaki Mizuno,
A dag-based algorithm for distributed mutual exclusion,
In International Conference on Distributed Computing Systems (ICDCS), 1991, 354_360.

NaT.87a Mohamed Naimi and Michel Trehel,
An improvement of the log(n) distributed algorithm for mutual exclusion,
In IEEE International Conference. On Distributed Computing Systems (ICDCS), 1987, 371-377.

NaT.87b Mohamed Naimi and Michel Trehel,
How to detect a failure and regenerate the token in the log(n) distributed algorithm for mutual exclusion,
Lecture Notes In Computer Science LNCS, 312 : 1987,155-166.

NaT.96 Mohamed Naimi and Michel Trehel,
A log(n) distributed mutual exclusion algorithm based on the path reversal,
Journal of Parallel and Distributed Computing (JDPC), 34 : 1996, 1-13.

Pra.17 Prabhu, B., Gajendran, E. and Balakumar, N.
Smart oil field management using wireless communication techniques,
IJIEST 2016, January–December, Vol. 2,

Rap.96 Rappaport, T.S.
Wireless Communications: Principles and Practice,
Prentice Hall ,PTR, New Jersey.

Ray.89 K.Raymond.
A Distributed Algorithm For Multiple Entries to a Critical Section,
Information Processing Letters 30, 1989, 189-193

Ray.91a Michel Raynal,
La communication et le temps dans les réseaux et les systèmes répartis,
Edition Eyrolles, 1991

Ray.91b Michel Raynal,
About logical clocks for distributed systems,
INRIA, Publication interne n°607, Octobre 1991

Ray.92 Michel Raynal,
Synchronisation et état global dans les systèmes répartis,
Edition Eyrolles, 1992

Ric.81 G. Ricart and A.K. Agrawala,
An Optimal Algorithm for Mutual Exclusion in ComputerNetworks,
Comm. ACM, Vol. 24, N°.1, Jan. 1981, 9-17.

San87 B.A. Sanders,
The informations structure of mutual exclusion algorithms,
ACM Transactions on Computer Systems, 5 : 1987, 284–299.

Sax.03 P. C. Saxena; Jagmohan Rai
A survey of permission-based distributed mutual exclusion algorithms,
Computer Standards & Interfaces, ISSN: 0920-5489, Vol: 25, Issue: 2, 2003, 159-181.

Sin.93 Mukesh Singhal,
A taxonomy of distributed mutual exclusion,
Journal of Parallel and Distributed Computing (JPDC), 18(1) : 1993, 94_101.

Sin92 M. Singhal,
A dynamic information structure for mutual exclusion algorithm for distributed systems.
IEEE Transactions on Parallel and Distributed Systems, 3(1) : 1992, 121_125.

Ska.85 I.Suzuki, and T.Kasami,
A distributed mutual exclusion algorithm,
ACM Trans. Computer Systems, Vol. 3, N°. 4, Nov. 1985, 344-349.

Sri. 92 P.K.Srimani, and R.L.N. Reddy,
Another distributed algorithm for multiple entries to a critical section,
Information Processing Letters 41, 1992, 51-57

Swa.07 Swaroop, A. and Singh, A.K.
A study of token-based algorithms for distributed mutual exclusion,
International Review on Computers and Software, Vol. 2, No. 4, 2007,302–311.

T.al.08 T Allaoui, M Yagoubi, M Djoudi, Y Ouinten,
A fast token based algorithm for multiple resources sharing in distributed systems,
IEEE International Conference on Innovations in Information Technology, 2008, 24-28

T.al.19 T Allaoui, MB Yagoubi, CA Kerrache, CT Calafate,
NFK: a novel fault-tolerant K-mutual exclusion algorithm for mobile and opportunistic ad hoc networks,
International Journal of Information and Communication Technology, 2019, 176-197.

Tam.12 Tamhane, S.A. and Kumar, M.
A token based distributed algorithm for supporting mutual exclusion in opportunistic networks,
Pervasive and Mobile Computing, Vol. 8, No. 5, 2012, 795–809.

Tan. 94 Andrew S. Tanenbum,
Distributed operating systems,
Prentice Hall, 1994

Thi.09 O.Thiare , M. Naimi ,
A Group *k*-Mutual Exclusion Algorithm for Mobile Ad Hoc Networks,
International Work-Conference on Artificial Neural Networks, 2009, 58-66

Vel.93 Martin G. Velazquez,
A survey of distributed mutual exclusion algorithms,
Technical report cs-93-116, Colorado State University, USA, 1993.

Wal.01 J. Walter, G. Cao, and M. Mohanty.
A k-mutual exclusion algorithm for ad hoc wireless networks,
Proceedings of the first annual Workshop on Principles of Mobile Computing (POMC 2001), 2001

Wal.97 J.E. Walter and S. Kini.
Mutual exclusion on multihop wireless networks,
Journal of Parallel and Distributed Computing, 1997, 25 :7–15.

Wal.98 J. Walter, J. Welch, and N.H. Vaidya,
A mutual exclusion algorithm for ad hoc mobile networks.
Dial M for Mobility Workshop, Dallas TX, 1998.

Wu.07 Wu, W., Cao, J. and Raynal, M.
A dual-token-based fault tolerant mutual exclusion algorithm for manets,
Third International Conference, MSN 2007, Mobile Ad-Hoc and Sensor Networks, Beijing, China, 2007,572–583.

Wu.08 Wu, W., Cao, J. and Yang, J.
A fault tolerant mutual exclusion algorithm for mobile ad hoc networks,
Pervasive and Mobile Computing, Vol. 4, No. 1, 2008, 139–160.

Yag. 97a Mohamed-Bachir Yagoubi,
L'exclusion mutuelle dans les systèmes informatiques répartis,
Thèse de doctorat de l'université d'Evry-Val-d'Essone, 1997

Yag.97b M.B. Yagoubi, A. Bouabdallah, J-C. König,
An improvement of O(logN) mutual exclusion algorithm to make it fault tolerant,
10th Int. Conf. On Parallel and Distributed systems, New Orleans (USA),October 1997.