A Totally Decentralized Document Sharing System for Mobile Ad Hoc Networks

Lionel BARRÈRE
LaBRI, Université Bordeaux 1
351 Cours de la Libération
33405 TALENCE Cedex
FRANCE
lionel.barrere@labri.fr

Arnaud CASTEIGTS
LaBRI, Université Bordeaux 1
351 Cours de la Libération
33405 TALENCE Cedex
FRANCE
arnaud.casteigts@labri.fr

Serge CHAUMETTE
LaBRI, Université Bordeaux 1
351 Cours de la Libération
33405 TALENCE Cedex
FRANCE
serge.chaumette@labri.fr

ABSTRACT

Many research projects have defined applications that are claimed to operate in a MANet environment. Consider for instance these software tools that make it possible to share information (files, data bases, etc.). It appears that most of the proposed solutions require a central server. Such a server makes it possible to store and then distribute a coherent copy of the data to the mobile units of the system, thus providing some sort of data consistency. This approach does not work in a real world MANet environment, where the presence of a central server cannot be guaranteed. We believe that the applications that really work in a MANet context are those that do not require global consistency. In this paper we present an application that we have designed and implemented and that makes it possible to share a document among the mobile nodes of a MANet. We furthermore decided to consider MANets without any specific networking layer, and to be more specific ***MANETs without routing***. The major originalities of our solution are that: -it works in ***a real MANet environment*** (nodes are really mobile and can be turned on or off at any time); - it makes it possible to modify the shared data ***without any central server***, still ensuring consistency. Of course the price to pay is a number of limitations that we describe.

Categories and Subject Descriptors:

I.m [Computing Methodologies]: MISCELLANEOUS; D.m [Software]: MISCELLANEOUS

General Terms:

Algorithms.

Keywords:

MANet, ad hoc networks, decentralized system, information sharing, mobile interactive application, mobility.

The work presented in this paper is carried out at LaBRI (Laboratoire Bordelais de Recherche en Informatique) and more precisely in the SOD (Systèmes et Objets Distribués) research team. It is partly achieved within the framework of the Sarah (Services Asynchrones pour Réseaux Ad Hoc) project supported by the ANR[1]. It is also partly supported by the DGA[2] by means of a PhD grant.

[1]Agence Nationale de la Recherche.

[2]Délégation Générale pour l'Armement.

MobiWac'06, October 2, 2006, Torremolinos, Malaga, Spain.

1. INTRODUCTION

Many research projects have defined applications that are claimed to operate in a MANet environment. Nevertheless, most of them require a central server. This does not work in a real MANet environment, where the reachability of such a central server cannot be guaranteed. Therefore the applications that really work in a MANet context are those that do not require the consistency among the pieces of data that they share to be dealt with in a centralized manner.

In this paper we present an application that makes it possible to share a document among the mobile units of a MANet, which we furthermore assume to be ***without any routing layer***. The major originalities of our solution are that: - it works in ***a real MANet environment***, i.e. where the mobile entities can be part of distinct unconnected groups; it makes it possible to modify shared data still ensuring consistency, ***without any central server***. From a practical point of view, a document can be splitted into pieces and shared among a set of mobile devices; the shared blocks can then be modified, splitted further, and the whole document can be locally rebuilt at each node depending on the effective encounters of the different mobile terminals. Each node is supplied with a tree based description of the shared document. When a block of data is splitted, the associated node is given two children nodes that carry the splitted contents. Copies of these blocks can then be given to other mobile nodes so that they can work on it. Of course, so that a block can effectively be shared, it is necessary to supply some sort of locking mechanism that will prevent multiple nodes from modifying their respective copies at the same time. A version number is also given to each instance of a block. These numbers are used when two nodes meet, so as to work out the most recent version of the block instances that they have in common. This makes it possible to synchronize them.

Of course some problems remain. For instance, it can be the case that a part of the global document is lost, if all the nodes that have an instance of this block of data leave the current network or simply crash. It can also be the case that a block of data, or more precisely all of its instances, remain locked forever for the same reasons. It will then be impossible to modify it further. Even though these situations are difficult to deal with, it nevertheless perfectly corresponds to what happens in the real life, for instance when you write a paper with colleagues, each one being in charge of a given section. These risks are inherent to the nature of really mobile applications.

2. OVERVIEW OF THE SYSTEM

The system that we have defined makes it possible to share a document among the nodes of a MANet. Several users can work (i.e. modify) at the same time on independent parts of the document. To work on a given part of a document, a node must acquire a copy or

instance of it. This results in multiple instances of a block, only one of which should be modifiable at a time. To ensure this property, a lock based mechanism is provided (see section 6). A synchronization process is also defined (in section 8) that uses version numbers to determine the most recent version among several instances of a block. When users meet, the different parts/versions of the document can then be synchronized.

2.1 Document representation

The shared document is structured as a binary tree. The contents of the document is carried by the leaves. An identifier is constructed for each block by concatenating either '1' for the left son or '2' for the right son to the identifier of its father.

2.2 Splitting and merging

This section defines how the document is shared and synchronized between nodes and what the implications of these operations are on the structure of the tree. We define two basic operations that are splitting and merging. The + symbol represents the concatenation of the contents (i.e the effective data) of two blocks.

Splitting

A part of the document, i.e. a block, can be splitted into pieces. This operation modifies the tree that describes the document as shown figure 1. We note (α, B_α) a node with identifier α and contents B_α.

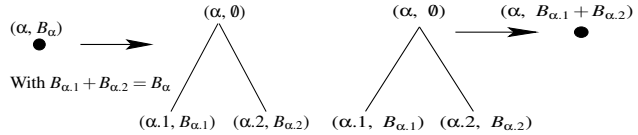

Figure 1: Splitting operation

Figure 2: Merging operation

Merging

In a symmetric manner, two blocks carried by two leaves of the tree can be merged into one single block. This operation modifies the structure of the tree as shown figure 2.

3. STATES

One of the major properties that we want to ensure is that each mobile node has the most recent version of the global document as possible, which of course depends on which other mobile nodes it has met. To achieve this goal each block has an associated lock, and each version of a block is given a version number. The lock is unique for a block and is shared by all the instances and hence versions of the block over the network. This lock has to be acquired by a node when it wants to modify its local instance of the corresponding block. The version counter attached to each block instance is used to distinguish between the different versions.

3.1 Notations

We have identified a number of attributes that are worth maintaining to describe the state of a block instance as seen by a given node. The full notation for a block instance is the extensive list of the values of these attributes:

(node, data, locked_by_me, available_for_me, version counter)

Because of space limitation this will not be described here and the reader is refered to [1] for a detailed explanation. In this section we focus on the reservation of the effective blocks of data. Only two attributes are necessary for this purpose and we will thus use the following short notation:

(locked_by_me, available_for_me)

The internal nodes of the tree are irrelevant in this section because nodes cannot be locked, the effective data being carried by leaves.

3.2 Possible states of a block instance as seen by a given node N

For a mobile node N its instance of a block B can be in the different states shown table 1. Note that the state (1, 1) does not make sense because a block cannot be lockable by a node that already locks it.

State	Is the lock in the same group as N ?	Who is locking the block ?
(0, 1)	YES	Nobody
(1, 0)	YES	N
(0, 0)	YES	Another member of the group of N
	NO	The block is locked or lockable in another group

Table 1: Possible states of a block instance at a given node N

3.3 Relationship between the states of the instances of a block within a group and between groups

The different states of the different instances of a block, as seen by different nodes at a given instant, are listed table 2.

Node	Other group members	Other groups
(0, 1)	(0, 1)	(0, 0)
(1, 0)	(0, 0)	(0, 0)
(0, 0)	(1, 0) for only one member, (0, 0) for the others or (0, 0)	(0, 0) (0, 0) in all other groups except in one group where either one node can be in state (1, 0) and the other nodes in state (0, 0) or all nodes are in state (0, 1)

Table 2: States of the instances of a block as seen by different nodes

4. TRANSITIONS

In this section we list the possible state transitions of a block instance. The state of a block instance changes when locking or releasing a lock on a (possibly other) instance of this block, or when a mobile unit joins or leaves a group.

4.1 Intragroup state changes

T1:	(0, 1) → (1, 0)	The node locks the block
T2:	(0, 1) → (0, 0)	The node leaves the group or another node in the group locks the block (T1)
T3:	(1, 0) → (0, 1)	The node releases the lock
T4:	(0, 0) → (0, 1)	A node in the same group as the current node releases the lock (T3) or a node with the block in state (0, 1) joins the group

4.2 State changes when a node joins a group

When a node joins a group, it can carry with it some block instances that it is locking. Thus, the join operation has the influence shown table 3. When a block is first created, it has a unique instance, and it is locked by the node that created it.

Joining entity	Other entities in the joined group
(1, 0), no change	no change
(0, 1), no change	(0, 0) → (0, 1)
(0, 0) → (0, 1) if state (0, 1) is in the group else no change	no change

Table 3: State changes when a node joins a group

Note that a node which is alone can have a block instance in state (0, 1) because in our system a single node forms a one-node group. The join operation supposes that a node is alone when it joins a group, i.e. it left its previous group before the operation.

4.3 State changes when a node leaves a group

When a node leaves its current group, it has to send a message to inform all the other members of the group that it is leaving. It then forms a new group that contains only itself. In table 4 we can see that when a mobile unit leaves a group, no other entity has to change the state of any of its block instances.

Leaving entity	Other entities in the left group
(0, 1) → (0, 0)	no change
(1, 0), no change	no change
(0, 0), no change	no change

Table 4: States changes when a node leaves a group

5. GROUP MANAGEMENT

In our system, mobile devices are organized in groups, and the assumption is that every member of a given group can communicate with all the other members of that group. In other words, the members of a group form a complete graph. This hypothesis would not be realistic in a general context but our target systems are small networks where a group is composed of nodes that are close to each other.

When first entering the system, a node is the only member of its group. It can thereafter join and leave other groups. When leaving a group, a node creates a new group of which it is the only member. Each group is attached a counter that is incremented each time its composition changes. The value of this counter is (re-)set to 0 when there is only one node in the group and each node keeps a history of the evolution of the group that it belongs to and a copy of the counter.

As explained above, the assumption is that groups are relatively stable, i.e that their composition does not change too quickly, and that the members of a group form a complete graph. Nevertheless this can still become false because of mobility or because a node is switched off. Therefore, a protocol has to be set up to make it possible to bring the system back to a configuration where the hypothesis is satisfied again. The protocol that we have designed is presented below.

5.1 Integrity checking protocol

The goal of the protocol that we define here is to check if all the members of the group of a given node are still reachable by this node, if the composition of the group has not changed relative to the knowledge that the node has of its group and if the version of the document that the node owns (including the state of block instances) is up-to-date relative to the group current version.

This process has to be applied every time a node is subject to or is achieving one of the two following critical operations:

- It is contacted by a node that wants to join the group. This supposes that the node is aware of the composition of its group at that instant so that it can check if the new candidate member is able to communicate with all the current group members (see section 5.2).
- It is willing to lock a block. Even though the locking process works in any configuration, it provides better results (i.e. no node wrongly believes that the block is locked - which is anyway without strong consequence thanks to the integrity checking protocol that we are defining) if the connectivity remains stable within the group while it operates (see section 6.1). This operation furthermore supposes that the node has an up-to-date version of the block to lock, since it would not make sense to modify an obsolete version of the document.

To meet these suppositions, a node does the following operations. It first checks that it can reach all the nodes of its group using some discovery mechanism and looking for the current version of the group composition. If it cannot reach some of its theoretical peers, it leaves the group. If this first step is successful, the node then checks the current version of the document present in the group and, if required, updates its own version. This especially includes the states of the nodes that wrongly believe that a block is locked (see section 6.1). The inconsistencies that make these update operations necessary can be the result of a loss of connectivity during block locking/releasing operations or group related operations. This will be pointed out in the following sections.

When this process is finished (at instant t) the node is sure to have a consistent and an up-to-date version of the document. At $t+\varepsilon$, if the node is still in the same group, either it has an up-to-date version of the document or if network failures occurred, it can get an up-to-date version running this protocol again. The network slow evolution assumption guarantees the convergence of this process.

5.2 Joining a group

A mobile node can join an existing group. This operation is so that once it is completed, the joining node has exactly the same version of the document as the other members of the group. In order to join a group, a mobile node D communicates with a node already in the group, say A. The process is as follows:

1. D informs A that it wants to enter its group. Assume that this group also currently contains B and C. Before proceeding, A checks its integrity using the protocol described in section 5.1.
2. A and D synchronize their versions of the document (this operation is described in section 8).
3. If D has the most recent version of some blocks, A broadcasts the synchronization information to all the other members of the group so that they can update their versions of the document.
4. D sends A the list of the mobile nodes that are in its communication range.
5. (a) If this list includes all the current members of the group (i.e. A, B and C), D can enter the group, that is now the set of nodes $\{A,B,C,D\}$. A increments the group version counter and notifies all the other members of the group that D is now a group member.

(b) If this list does not include all the current members of the group, then *D* is denied access to the group.

6. *A* sends *D* all the changes that happened in the group since step (2) (block updates, ...).

It is easy to see that during the first three steps, group operations have no influence over the joining process. Difficulties can happen during steps 4 and 5, if the group accepts a new member after *A* has sent the list of mobile terminals that it can reach, and if this list does not include the new member. In this case, that should anyway rarely happen, *D* is denied access to the group, but it can still retry to join later. Note that step 6 may also lead to an inconsistent state in case of loss of connectivity that would prevent a node from being informed. This will be solved by the integrity checking process of section 5.1.

5.3 Leaving a group

A node that wants to leave the group that it belongs to has to inform all the other members. The message that it sends for this purpose does not contain any information about the blocks on which it keeps a lock because this operation has no influence over the state of the instances of this block owned by the other nodes of the group (see table 4). In case a node cannot be informed this will again be solved by the integrity checking process of section 5.1.

6. CONSISTENCY MANAGEMENT

In this section we explain how we ensure data consistency in our system by using locks on blocks.

6.1 Locking a block

Locking a piece of data is a difficult operation in a MANet because of the network versatility. Before locking a block a node needs to check the integrity of its environment as described in section 5.1. By doing so we ensure that this node has the most recent version of the document and that it is in the communication range of all the other members of its group. For the sake of explanation, we first assume that the connectivity inside the group does not change during the block locking process. The case when the connectivity changes will be specifically dealt with right after.

A node N that wants to lock a block asks all its group peers if it can do so, and collects the number of positive answers that it gets. More precisely, it proceeds as follows:

1. N sends a request to each (one at a time) other member of its group. The message contains the block identifier, its version number and a unique identifier of the initiating node, for instance its IP or MAC address.

2. The other nodes in the group can answer as follows:

   - *OK_[ID]_[Num_of_Positive_Answers_obtained]*. The answering node M agrees that N gets the lock. M will then put the block in the state (0, 0). This means that M considers that the block will eventually be locked by N and is therefore not available for itself. This may be be false if N does not get the lock, but this way to proceed makes it possible to avoid an additional message from N (and the associated connectivity loss problems) to inform the other nodes that it has got the lock (or not). To get back to the interpretation of the answer sent by M, there are two possibilities:
     - M does not want the lock on block B.
     - M wants to lock the block B but it has an ID smaller than N. M then sends N the number of positive answers its has cumulated, and N adds this number to its own total.
   - *No_InvalidVersion*. N has an invalid version of the block B and must update it.
   - *No_[ID]*. The answering node is also willing to lock the block at the same time and its ID is bigger than the ID of N. N then sends it the number of positive answers that it has already obtained.

3. If N gets a number of positive answers bigger than half the number of members of the group, it considers that it has obtained the lock on the block. This number must be strictly bigger than the integer part of $(n+1)/2$, n being the number of other nodes of the group, minus node N. The lock is thus unique because only one node can obtain $(n+1)/2$ positive answers.

As soon as a node willing to lock B gives a positive answer to another node, what means that its ID is lower than the ID of the other node, it stops asking for the lock. It then considers that the block is not available for it any longer even though this might be false (see above), and puts it in the state (0, 0). Such an erroneous state will be corrected by the integrity checking process of section 5.1.

6.2 Unlocking a block

To release a block B, a node N increments the version counter associated with its instance of B and sends a message to all the other members of its group. This message contains the updated block, including the incremented version counter. By doing so, update information are systematically propagated in the group. In case a node cannot be informed the integrity checking process of section 5.1 will again solve the problem.

Blocks that become unlockable

If a technical problem occurs, it may happen that the node that locks a block B becomes unavailable (because it has been switched off or has crashed or has moved away).The consequence is that the block is then unlockable forever and it will thus be impossible to modify it again. This is one of the drawbacks of a totally decentralized approach.

7. INFLUENCE OF SPLITTING AND MERGING OVER VERSION COUNTERS

The version counter is evolved so that the most recent version of a block has the biggest counter value. The combination of the operations described bellow with the use of a lock as described in section 6 makes it possible to guarantee this property.

Splitting.

Splitting a block has no influence over its version counter. The newly created sub-blocks are given version number 0 and they are locked.

Merging.

When merging two blocks the version counter of the father block is incremented and the sub-blocks are dismissed. The resulting block is locked by the node achieving the operation.

8. SYNCHRONIZATION

The synchronization process that occurs between two nodes A and B is the operation that consists in building the most recent version of the document based on the versions of A and B. This operation typically occurs when a node joins a group (see section 5.2, step 2).

Most recent version of a block

The following rules are used to work out the most recent version of a block among two instances:

1. The bigger the version counter, the more recent the block.
2. If a node instance and a leaf instance have the same version counter then the version of the node is more recent than the version carried by the leaf. This is due to the fact that the locking process guarantees that only the most recent version of a block can be locked and thus splitted.
3. If two instances of a block have the same version counter and are in the same state then the two versions are the same.

Synchronization

Two nodes that want to synchronize first compare their versions of the tree that both describes the document and carries its contents. To prepare this operation they exchange a representation of a prefix ordering depth first search of the tree. Based on the result of the comparison, they exchange the blocks that have been identified as being different.

9. RELATED WORK

The domain of MANets is very active. A lot of projects are being developed, ranging from new routing protocols to high level applications. In this section we focus on applications that deal with sharing/disseminating information.

One of the major categories of applications on MANets are file systems. Several approaches can be found in the literature. AdHocFS [2] is such a system developed at INRIA[3]; this system is based on the use of groups of mobile devices. All groups can synchronize the versions of the files on a central server. Within each group only one node can modify a particular block of a file at a time, based on a token management mechanism. Another example of a distributed file system is MFS [3]. It seems that file systems are hard to deal with within the context of MANets because they need stability to ensure data consistency. For example AdHocFS uses a central server to ensure consistency between distinct ad hoc user groups. Another important class of MANet applications that address information sharing are those dealing with the dissemination of information (see for instance [4, 5, 6, 7]). The major difficulty of these systems is to ensure that the maximum number of mobile terminals will eventually get the files they want. There is no issue of consistency here because the files to disseminate cannot usually be modified.

Another important domain targeted by MANets is that of support tools for rescue teams. Cenwits [8] is such a tool. This application makes it possible to disseminate the location of hikers so as to evaluate their position when they need rescue. This system uses a fully decentralized view and opportunistic connectivity between hikers or between a hiker and an access point that transmits information to a central server. Workpad [9] is another system that helps organizing collaborative work of rescue teams. Its is supported by a two levels architecture, a back-end peer-to-peer network and a front-end composed of groups of connected mobile devices. Each group contains a coordinator that is the relay between the front-end and the back-end. At the front-end level, the system does not really propose any connectivity management solution within groups: nodes are asked to move to maintain connectivity. When a node is going to be disconnected another node is elected to follow it, becoming a bridge and a multi-hops model is then used. There is nothing to deal with the apparition of unconnected groups, the assumption being that this does not happen in the target context.

[3]Institut National de Recherche en Informatique et en Automatique.

10. CONCLUSION

In this paper we have presented a system to share a document in a fully distributed manner among the nodes of a mobile ad hoc network. We have explained how we organize the parts of the document as a binary tree. We have shown that by using version counters and locks on the copies or instances of the blocks of the document, we can ensure data consistency and guarantee that a node always has the most recent version of the document as possible. Our contribution is thus an architecture that makes it possible to share and modify a document in a totally decentralized and mobile manner, still ensuring data consistency. We have an effective implementation of this system that shows its feasibility. It uses the Java[4] technology and Wi-Fi 802.11. It runs on Linux Debian platforms. This application is a prototype and we currently use the serialization mechanism of Java to transmit block instances between nodes. We plan to use XML in the next version. This work is currently being extended in collaboration with the DGA to the management and collection of information by troopers on the battle field, the shared document being a strategic map. In this context evaluations, will be achieved regarding some critical aspects such as synchronization duration and data availability, based on the moves of the mobile stations.

11. REFERENCES

[1] Lionel Barrère, Arnaud Casteigts, and Serge Chaumette. A Totally Decentralized Document Sharing System for Mobile Ad Hoc Networks. Technical report, LaBRI, Université Bordeaux 1, FRANCE, 2006.

[2] Malika Boulkenafed, Valérie Issarny, and David Menté. AdHocFS : A serverless file system for mobile users. Technical Report RR-4303, INRIA, 2001.

[3] Benjamin Atkin and Kenneth P.Birman. MFS: an adaptive distributed file system for mobile hosts. Technical report, Departement of Computer Science Cornell University, Ithaca, 2003.

[4] Hervé Roussain and Frédéric Guidec. Dissémination asynchrone d'information en mode peer-to-peer dans les réseaux ad hoc. In *Proceedings of the first french-speaking conference on mobility and ubiquity, Ubimob'04*, 2004. Nice, France.

[5] Siddhartha K.Goel, Manish Singh, Dongyan Xu, and Baochun Li. Efficient peer-to-peer data dissemination in mobile ad-hoc networks. In *Proceedings of International Workshop on Ad Hoc Networking, IWAHN*, August 2002. Vancouver, Canada.

[6] Maria Papadopouli and Henning Schulzrinne. Design and implementation of a peer-to-peer data dissemination and prefetching tool for mobile users. In *Proceedings of the 2nd ACM International Workshop on Modeling and Simulation of Wireless and Mobile Systems, MSWIM*, 1999. Seattle, USA,.

[7] Maria Papadopouli and Henning Schulzrinne. Seven degrees of separation in mobile ad hoc networks. In *Proceedings of the IEEE Globecom '00'*, 2000. San Francisco, USA.

[8] Jyh-How Huang, Saqib Amjad, and Shivakant Mishra. Cenwits: A sensor-based loosely coupled search and rescue system using witnesses. In *Proceedings of SenSys'05*. ACM, 2005. San Diego, USA.

[9] Massimo Mecella, Tiziana Catarci, Michele Angelaccio, Berta Buttazzi, Alenka Krek, Schahram Dustdar, and Guido Vetere. Workpad: an adaptive peer-to-peer software infrastructure for supporting collaborative work of human operatoes in emergency/disaster scenarios. In *Proceedings of the 2006 International Symposium on Collaborative Technologies and Systems*. IEEE Computer Society, 2006. Las Vegas, USA.

[4]Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. The authors are independent of Sun Microsystems, Inc. The other marks are the property of their respective owner.