UNIVERSITY OF OSLO
Department of Informatics

Towards Secure and Reliable Information Sharing in Emergency and Rescue Operations

PhD thesis

Matija Pužar

July 2nd 2010

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 978

ISSN 1501-7710

Cover: Inger Sandved Anfinsen.
Printed in Norway: AiT e-dit AS.

Produced in co-operation with Unipub.
The thesis is produced by Unipub merely in connection with the
thesis defence. Kindly direct all inquiries regarding the thesis to the copyright
holder or the unit which grants the doctorate.

Abstract

Efficient information sharing among rescue personnel is crucial for a successful rescue operation. If computer networks were actively used by the rescue personnel, it would allow for more efficient communication, and information sharing in general, compared to standard walkie-talkies still in use today (often more than one at a time). However, emergency and rescue operations present the system with a number of characteristic requirements, compared to traditional networks. The most significant difference is that the network must be built up by the rescue personnel on the spot, and that the presence of existing infrastructure cannot be relied on. Moreover, the rescue personnel's movements make the network topology very dynamic and unstable. We call this type of networks mobile ad-hoc networks (MANETs).

In this thesis, we analyze in detail the requirements for usage of MANETs in emergency and rescue operations, with particular focus on security and data sharing. Based on this analysis, the thesis suggests solutions to a selected set of major challenges. The thesis' main contributions are threefold.

The first contribution is a concrete solution for one of the main security issues, namely ensuring that only authorized personnel has access to the network. This is achieved by means of a simple and efficient key exchange protocol that relies on device credentials being installed prior to the operation.

Next, the thesis contributes with a shared data space that can be used to efficiently and robustly distribute information among the rescue personnel. The distributed nature of the data space is transparent to the applications, as well as the fact that data versioning is performed for the purposes of consistency, conflict resolution, and auditing. Data placement in such a distributed and dynamic environment has to be performed with outmost care. In the cases where network topology or the applications' access pattern to the data space are unknown upfront, we show that placing replicas on 10 % well chosen nodes, achieved e.g. by means of clustering techniques, leads to close-to-optimal placement with regards to network usage.

The final contribution of this thesis is a network emulation test-bed implemented to facilitate development of specialized applications and protocols for MANETs. The test-bed has been utilized in a number of Master's and PhD theses, demonstrating its usefulness and flexibility with respect to development time and cost, as well as choice of programming languages.

Acknowledgment

First and foremost, I would like to thank my supervisors Thomas Plagemann, Jon Andersson and Yves Roudier for most valuable inputs and guidelines throughout these years. The same goes to all of my current and former colleagues from the Distributed Multimedia Systems group at the Department of Informatics, especially my fellow PhD students from the Ad-Hoc InfoWare project, Katrine Stemland Skjelsvik, Ovidiu Valentin Drugan and Norun Christine Sanderson.

I would also like to thank my former supervisor Maja Matijašević from the Faculty of Electrical Engineering and Computing in Zagreb, for showing me the way.

From the MIDAS project, there are a few persons I would particularly like to thank for a great collaboration in our technical discussions, both during and after the project's lifetime – José Pablo Gañán Blanco, Michał Koziuk, Alisa Devlić and Wybe Horsman.

Special thanks to Ellen Munthe-Kaas and Katrine Stemland Skjelsvik for taking their time to proof-read the thesis and giving me valuable feedback, and to the latter, for the crash course in Norwegian during my first years in the country, and for not giving up on me or kicking me out of the office.

Finally, big thanks to my wonderful wife Snežana for her support and for being who she is. 😉

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AODV	Ad-hoc On-Demand Distance Vector Routing
API	Application Programming Interface
DBMS	Database Management System
DENS	Distributed Event Notification Service
DSDV	Destination-Sequenced Distance Vector routing
DSR	Dynamic Source Routing
CPU	Central Processing Unit
CRT	Communication and Routing
DA	Data Allocator
DS	Data Synchronizer
GMDM	Global MetaData Manager
GPS	Global Positioning System
GPRS	General Packet Radio Service
HNA	Host and Network Association
LS	Local Storage
MANET	Mobile Ad-hoc Network
MDS	MIDAS Data Space
MIDAS	Middleware Platform for Developing and Deploying Advanced Mobile Services
NEMAN	Network Emulator for Mobile Ad-hoc Networks
OLSR	Optimized Link State Routing protocol
PDA	Personal Digital Assistant
QA	Query Analyzer
SKiMPy	Simple Key Management Protocol
SM	Subscription Manager
SQL	Structured Query Language
TC	Traffic Controller

Part I

Overview

Chapter 1

Introduction

In an emergency and rescue operation, where people's lives might be at stake, efficient data sharing between the organizations and individuals involved is of outmost importance for the mission's success. Disasters often happen where and when it is least expected, making it hard or impossible to prepare a working infrastructure on the location in question, or to expect such an infrastructure to be present when the rescue personnel arrives. For that reason, a more practical solution needs to be present, which would work anywhere and at any time.

A mobile ad-hoc network (MANET) is a wireless network created on the spot by the devices present at the time. Devices taking part in the network's activity might be of very heterogeneous nature with regards to hardware and software specifications. They do however need to support the same network protocols to be able to form a network. In a MANET, each device works both as a host node and a router node, and routes between nodes are constantly updated depending on the nodes' connectivity. Existing infrastructure, if present, can be used to increase connectivity or to gain access to external networks (such as the Internet), but it is not a prerequisite. Given the abovementioned characteristics, MANETs are a natural candidate for our target scenario, i.e., emergency and rescue operations.

However, using MANETs introduces challenges on several fronts. The broadcast nature of wireless networks has a big impact on network and data security. On the other hand, the dynamic and unpredictable nature of MANETs influences in a high degree data availability. Both of these areas need to be researched, and satisfactory solutions need to be present, in order to achieve a system that is acceptable for data sharing in emergency and rescue operations.

This thesis is tied to two research projects, Ad-Hoc InfoWare [39] and MIDAS [9], both of which focus on MANETs in emergency and rescue operations. As part of these two projects, the thesis focuses on securing the network layer and implementing and optimizing the presented solution for a distributed data sharing space. Furthermore, the thesis contributes with a working emulation environment, which was necessary to develop, test, and evaluate applications and protocols.

1.1 Problem Statement

Emergency and rescue operations present the system with numerous requirements. In this chapter, we present the main requirements of interest for this thesis.

The first, basic requirement is security. A typical example are patients' medical records that might be sent through the network, where one must make sure that only certain people (e.g., medical personnel) are allowed to see them. On the other hand, information such as pictures and other details of the incident area might be of interest for all the involved parts, which clearly shows that different security policies need to be present with regards to access control and confidentiality.

Authorization is another security objective that needs to be taken care of. Given the fact that malicious individuals or groups not only might be present at the incident area, but might also be directly responsible for the incident itself, it must be ensured that only authorized persons have access to the network. A malicious person could easily interfere with the rescue operation by means of signal jamming or by inducing false routing information, which would in both cases disrupt the network and make data sharing over MANETs impossible. Signal jamming is something that needs to be addressed by other means, e.g., channel coding at the link layer [28], or by detecting at the physical layer, physically locating the source, and making sure it is disabled. On the higher levels, however, more advanced methods need to be designed and implemented to protect the network from intruders. It has been shown that a very small percentage of misbehaving nodes might cause the whole network to collapse [16]. Consequently, protecting the routing protocol is fundamental for having an operational network.

Availability of data is also important for the success of a rescue operation. In traditional wired networks, one can assume that the data will always be available, except in specific situations. In mobile networks, however, a node hosting important data might suddenly disappear due to the device entering a building, its battery being empty, etc. Thus, such situations constitute more a rule than an exception, making it necessary to have mechanisms in place that would prevent data loss and increase availability. This can only be achieved by means of data replication. Replication, in cases where data is constantly generated or updated, introduces the need for synchronization, which is costly with regards to network usage. The number of replicas, as well as their locations in the network, should therefore be planned and implemented carefully. Failure to do so might cause synchronization traffic to consume all network resources, and thus render the whole infrastructure unusable to the rescue personnel.

To ensure that the applications and protocols work correctly and meet the expectations imposed in the design phase, they need to be tested and evaluated. For network protocols specifically, there are three main possibilities to do so, namely *simulation*, *emulation*, and *field tests*. *Simulations* usually do not run in real-time, and are suitable for larger networks. In addition, they might include a detailed representation of the physical layer, and the results are reproducible. The downside is that one must spend a certain amount of time to learn the simulator, and implement the protocol specifically for a given simulation, only to be re-implemented later for real devices. *Emulations* allow for a certain part of the protocol stack to be simulated, while the rest consists of real code (implementation) running in real-time. This can considerably reduce the time needed to implement an application or protocol, with the possibility to later install them to target devices with minor changes or no changes at all. Emulation results can be reproducible to a certain extent. The downside is that the emulation environment, depending on its implementation, might not be scalable. *Field tests*, where real code runs on real devices, give the most realistic results. They are, however, the most costly alternative with regards to the number of devices and people involved. In addition, debugging and bug-fixing can be problematic, and results are hardly reproducible.

To summarize, this thesis addresses the following problems:

- security in MANETs,
- data sharing and replica placement in MANETs, and
- testing and evaluation of MANET protocols.

1.2 Methodology

To address the abovementioned problems, we used the following methods. We performed an in-depth requirement analysis of the target scenario, i.e., emergency and rescue operations. We analyzed the existing related work and state-of-the-art in the topics in question, namely security in MANETs, data sharing and replica placement in MANETs, and network simulators and emulators. Since we aim for a real proof-of-concept implementation, we follow a typical *systems approach* comprising design of applications and protocols, followed by their implementation in various programming languages, and last by testing and evaluation. For the purpose of testing and evaluation, we developed our own emulation platform, called NEMAN, which is also one of the contributions of this thesis. We designed and ran most of the experiments on the emulation platform, but for more resource demanding experiments, we used modeling and simulation. We successfully ran some of the solutions in field tests on target devices.

1.3 Main Contributions

The main contributions of this thesis can be summarized by the following four claims:

Claim 1:

In emergency and rescue operations, security has to be present at the lowest layers, in order to prevent malicious nodes from disrupting the network. The hierarchical organization of the entities involved in these kinds of operations can provide a basis for trust establishment between rescue personnel's devices. We provide a solution that exploits this fact in order to agree on a shared key. The solution can be used to ensure that intruders cannot disrupt the network by injecting false routing information. Moreover, the solution can be used to secure all the traffic from being tampered with by unauthorized persons.

Claim 2:
With a secured infrastructure, the system can be used for information sharing between the rescue personnel. A shared data space, such as MIDAS Data Space, can be used to efficiently and robustly distribute information, even on small resource-limited devices. The strict requirement of accounting for such operations can be achieved by means of versioning.

Claim 3:
It is possible to achieve a close-to-optimal placement of table replicas within the network, which works well for most scenarios where the applications' access pattern to the data space or the network topology are unknown a priori. At a later point, the system can adjust to the concrete situation by analyzing the applications' access pattern and network topology.

Claim 4:
In order to develop and evaluate protocols, an adequate test-bed has to be in place. An emulation test-bed is the most flexible choice with respect to development time and costs, as well as choice of programming languages. We have implemented such an emulation test-bed and used it to develop, test and evaluate a variety of protocols.

1.4 Structure of the Thesis

This thesis is divided in two main parts – the introductory part and the collection of research papers.

The introductory part gives some background information about the work presented in the papers. In Chapter 2, we present the main issues in mobile ad-hoc networks that the thesis focuses on, as well as the two research projects in context of which the thesis is done. Chapter 3 describes the application requirements for the two research projects, and consequently for this thesis. In Chapter 4, we introduce the MIDAS Data Space and the proof-of-concept implementation made as part of the MIDAS project. Chapter 5 gives a short overview of the research papers included in the second part. Finally, in Chapter 6 we give a conclusion, including critical review of the claims introduced in Chapter 1.3, and plans for future work.

The second part (Chapters 7-13) presents the seven research papers that constitute the main contribution of the thesis. Each paper is given in its entirety, as published in the respective conference or workshop. The papers have, however, been re-formatted to match the rest of the thesis' layout.

Chapter 2

Background

In this chapter, we introduce mobile ad-hoc networks (MANETs) and give an overview of the main topics within MANETs that constitute the background for work on this thesis, namely security and data sharing. We also present briefly the research projects this thesis is tied to.

2.1 Mobile Ad-Hoc Networks

While in traditional infrastructured networks the topology is static and predictable, and nodes often have distinct roles (end-user node, router, access point, etc.), mobile ad-hoc networks (MANETs) are basically characterized by having each node performing all these functions. Another important difference between infrastructured networks and MANETs is that the former often have a steady topology, while the latter's main characteristic is that their topology can be highly dynamic and unpredictable. As nodes move around, they constantly form new links, while removing old ones.

The advantage of MANETs is that one can establish them at any place and any point of time, as they are not dependent on any infrastructure being set up upfront. If an infrastructure is present, however, MANETs can easily connect to it to achieve a broader range. As an example, a node in a MANET can have an additional 3G/GPRS/satellite connection and use it to act as a gateway to the Internet, and through it to the organization's headquarters. Another example might be to use such a connection to communicate with another MANET on a distant or out-of-reach location, e.g. two rescue teams located on the opposite sides of a tunnel.

As such, MANETs represent ideal candidates for emergency and rescue operations, whose locations are not known upfront, or military operations on an enemy territory, whose locations might be known upfront but there is no physical access to them.

2.1.1 Routing in MANETs

To be able to communicate and to forward packets within a MANET, nodes in the network must use a routing protocol. Routing protocols in MANETs can be roughly put into two categories, *proactive* and *reactive*.

In proactive routing protocols (such as OLSR [4], DSDV [36]), nodes regularly send broadcast beacons to inform neighbors about their presence. As a consequence, each node can at any time have a complete picture of the network's topology, but having a proactive routing protocol causes constant traffic that might be unnecessary.

Reactive protocols (such as AODV [35], DSR [23]), on the other hand, are triggered by outgoing traffic, and routes are only discovered when needed. As a consequence, nodes might not know about their neighbors until they in fact try to communicate with them, but no bandwidth is used by the routing protocol unless necessary.

A variety of other routing protocols exist for MANETs, such as hybrid protocols trying to include the best characteristics of both proactive and reactive ideas, geographical routing protocols, taking into consideration nodes' physical locations, multicast protocols, etc.

As part of the research projects this thesis is bound to, the choice fell on the proactive principle, more precisely the OLSR protocol. Using a proactive routing protocol opens doors to a number of possible applications that rely on (or benefit from) knowledge of the network topology. One such application, described in this thesis, is using this knowledge to better plan the location of database replicas in the network, to minimize network traffic caused by remote database queries and data synchronization, and to increase the probability of data being available in case of network disruptions. The Distributed Event Notification Service (DENS [52]) is another example of an application that benefits from knowledge of the network topology.

The OLSR Protocol

OLSR stands for *Optimized Link State Routing*, and is a proactive routing protocol for MANETs. OLSR aims at providing each node in the network with the whole logical picture of what the network looks like (i.e., network topology). This is mainly achieved by means of different types of messages.

Each node sends periodic *Hello* messages (typically every 1-2 seconds), to inform potential neighbors of its presence. Hello messages include a list of all known direct (i.e., 1-hop) neighbors, making it possible for recipients to create a picture of the whole 2-hop neighborhood. By doing this, it is easy for a node to find the minimal number of its 1-hop neighbors needed to cover the 2-hop neighborhood, for the purpose of flooding the network. These chosen nodes are called *Multipoint Relays* (MPRs) and are specific to each node separately, and each node's MPR set is known to all of its neighbors. This way, each neighbor knows whether it should later retransmit broadcast messages or not. *Topology Control* messages are an example of such broadcast messages, which include all the information necessary to calculate routes.

OLSR allows for connection to other networks as well. By means of *Host and Network Association* (HNA) messages, a node can inform other nodes in the MANET if it has additional interfaces which might give connectivity to other networks.

OLSR has been designed with modularity in mind. The packet format allows for multiple messages being sent together, as well as including support for custom messages. As part of our research, we have implemented an extension to the security plugin [15] distributed with the UniK olsrd [58] implementation, adding the functionality of node authentication and key distribution, as described in Chapter 8 (i.e., Paper #2). We used the same functionality to work on other plugins, such as transparent gateways [2], piggybacking of custom messages on broadcast packets [54], etc.

2.1.2 Security in MANETs

Use of wireless networking presents the system with a series of security related challenges. Network security comprises four main objectives: *authenticity*, *integrity*, *access control,* and *non-repudiation*. Starting from the medium itself, one of its main characteristics is that anyone with a tuned wireless device can listen to the traffic in the network, as well as generate and induce bogus traffic into it. Since neither of these two issues can be efficiently prevented in a rescue operation, they introduce problems into at least three of the abovementioned security categories:

- Authenticity: mechanisms need to be present to ensure that only authorized devices can be part of the network.
- Integrity: it must be ensured that no one can manipulate the network traffic by, e.g., changing its contents before retransmitting it.
- Access Control: some data must be kept confidential, i.e., inaccessible to unauthorized recipients.

Another issue not directly related to data security but still directly related to the efficiency of the network (and as such to the efficiency of the rescue operation itself) is signal jamming. Unlike wired networks where a device needs physical access to the medium to perform any action, wireless networks are highly prone to such attacks and there is no real solution other than locating the perpetrator and disabling the source of jamming, or at least by using some channel coding techniques at the link layer [28].

All the abovementioned issues are considered as *external* attacks [24], i.e., attacks coming from nodes that are not (or not supposed to be) part of the network. One of this thesis' main contributions, presented in Chapter 8 (i.e., Paper #2) is a key management protocol that can be used by the rescue personnel to prevent unauthorized nodes from joining the network.

A much different, and in some cases more dangerous type of attacks are so called *internal* attacks [46]. These attacks come from nodes that have already been authenticated and as such considered legitimate members of the network, but at some point have become compromised. This can happen if devices are stolen, lost and then found by a malicious person, or even worse, if their legitimate owner becomes compromised. Detecting and excluding such nodes is a much more difficult process and requires specialized solutions.

Other than attacks, the system must take care of confidentiality within the network. Inter-organizational collaboration is one of the key functionalities in a rescue operation. Nevertheless, different organizations may have different security requirements and policies and, in addition, not all data within the network should be seen by every member. Examples of such data may be medical records, police records, other personal or confidential information, etc. Additional challenges are imposed by different organizational structures and levels of confidentiality within the organizations themselves, which could also change dynamically.

With regards to implementation, security can be implemented in either the traditional layered approach or a more adaptive cross-layered one, both having their advantages and disadvantages. The layered approach, by putting clear borders to data flow, offers a high level of security. In our case, a more flexible approach might be a better choice, provided that it does not significantly weaken the overall security level. A lightweight adaptable cross-layer middleware solution, based for example on the reflection technique [29], [3], would allow middleware services to adapt to the heterogeneous dynamic environment. Examples of such architectures are Open ORB 2 [3] and ReMMoC [14]. The programming language Obol [1] can be used to face the security issues, and with help of Aspect Oriented Programming [26], the cross-concern integration can be faced early in the development phase.

For a more extensive look into security issues in MANETs, focused on middleware for emergency and rescue operations, see Chapter 7 (i.e., Paper #1).

2.1.3 Data Sharing and Replication in MANETs

The dynamic and unpredictable nature of MANETs poses special challenges with regards to data sharing, more precisely availability and consistency. Availability can be degraded by nodes disappearing, something that can be compensated by means of data replication. Data replication is, however, a potentially costly operation with regards to the amount of data that need to be transferred from one replica to the other, in order to achieve a consistent state where all replicas have the same information. Such a consistent state can only be guaranteed through the use of transactions. Due to MANETs being prone to partitioning and frequent route changes, transactions might often fail or lead to locks, and are therefore undesirable. Since it has been proven that it is impossible to have a system that is prone to partitions, and that at the same time provides data consistency and availability [8], we must accept a trade-off between the levels of data consistency and availability. *Eventual consistency* is such a model where, under certain requirements, it is guaranteed that all accesses to the data eventually will give the same result. The MIDAS Data Space, described in Chapter 4, is an example of a system that ensures eventual consistency by means of *eager* and *lazy synchronization*.

A variety of other data replication techniques exist for databases in MANETs, some of which are categorized and presented in [34].

2.1.4 Testing and Evaluation of MANET protocols

Testing and evaluation are important stages in the process of developing network applications and protocols. For MANETs, this is particularly difficult due to the fact that the network topology might be constantly changing. Applications and protocols need to be designed to cope with this issue, but whether they actually manage it is something that needs to be tested. Three basic methods are available for doing so, namely *simulation*, *emulation*, and *field tests*.

In simulations, the whole protocol stack is simulated, making it necessary to implement everything specifically for the given simulation environment. It is the cheapest solution with regards to the number of devices needed, and results are reproducible. However, the learning curve for the simulator may be steep, and applications need later to be rewritten in order to run on real devices. Network Simulator ns-2 [55], its successor ns-3 [57], OMNeT++ [60] and GloMoSim [64] are examples of well-known simulators used within the research community.

In field tests, where real devices are used, real implementations of the applications are also used. This may, however, be expensive with regards to both manpower and the number of devices needed. Moreover, deployment and debugging can be a challenge, and the results (which might depend on persons' movements) are not reproducible.

Emulation environments present a compromise between simulation and field tests, by having parts of the protocol stack (typically the lowest layers) being simulated, while other parts run the real implementation. This is the most cost-effective way of testing and evaluating network applications and protocols since not every node needs its own physical device, and the code can later be moved to real devices with minor changes or no changes at all. NEMAN [42], presented in Chapter 9 of this thesis, and a distributed version of NEMAN called DINEMO [13] are examples of network emulators. IMUNES [62] with its follow-up project VIRTNET [61], Mobile Emulab [21], MobiNet [30], and ORBIT [47] are some other examples. ns-3, even though it is mainly a simulator, introduces support for different types of emulation, with respect to which layers are being simulated or emulated.

For a comprehensive list of emulation platforms and test beds, see the surveys presented in [12] and [25].

2.2 Research Projects

Work on this thesis has been done in the context of two research projects, Ad-Hoc InfoWare and MIDAS. Both the projects have emergency and rescue operations as a target scenario, though they focus on different aspects of the scenario and approach it in different ways. In this chapter, we describe both the projects individually, as well as their connection to this thesis.

2.2.1 Ad-Hoc InfoWare

Ad-Hoc InfoWare [39] is a research project funded by the Norwegian Research Council, running in the period 2003-2006. The aim of the project is to develop middleware services for emergency and rescue operations, in order to fulfill the following requirements: Intra- and inter-organizational information flow, service availability, context management, profiling and personalization, group- and organizational support, dynamic security, communication, resource sharing (especially data sharing), and graceful degradation. These requirements are addressed by six middleware concerns:

- *Knowledge Management* – to handle ontologies, support metadata integration and interpretation;
- *Context Management* – to manage context models, context sharing, profiling and personalization;
- *Data Management* – to cater for capabilities similar to those of distributed databases;
- *Communication Infrastructure* – for supporting distributed event notification, publish and subscribe services, and message mediation;
- *Resource Management* – to register and discover information sources and web services as well as resources available, to handle neighbor awareness, computation and application sharing, mobile agents, proxy and replica placement, and movement prediction;
- *Security Management* – for access control, message signing and encryption, supporting group- and organizational structure, group key assignment, and dynamic security services.

In the middleware architecture, these concerns correspond to five components which we present shortly in this chapter, together with the respective PhD theses addressing them.

The *Knowledge Manager* component corresponds to the concerns Knowledge Management and Context Management. The purpose of this component is to provide flexible services that allow relating metadata descriptions of information items to a semantic context and support management of knowledge sharing and integration in a rescue operation scenario. The Knowledge Manager offers support for the dissemination, sharing and interpretation of ontologies, and browsing and querying of ontologies and ontology contents. Issues that needed addressing included understanding across domains and organizations through use of knowledge management techniques, avoiding information overflow through content filtering and personalization, managing availability of information, metadata and ontologies, offering information query and retrieval services, and supporting information exchange. Work on the *Knowledge Manager* has been documented as part of Norun Christine Sanderson's PhD thesis [49]. The thesis has contributed with a three-layered architecture for efficient metadata management, an approach to ontology based dynamic update, and a high level design of the Knowledge Manager component targeted at the given scenario's requirements.

Distributed Event Notification Service (DENS) and *Watchdogs* provide a publish/subscribe service and delay-tolerant delivery of notifications in case of, e.g., network partitioning. In the publish/subscribe service a subscriber subscribes for information, and the publishers publish information, independently. If DENS cannot deliver a notification to a subscriber, the service will store the notification and try to deliver the notification using the store-carry-forward paradigm. The main design goals for DENS were to support a flexible subscription model, ensure a high delivery ratio, as close to at-least-once semantics as possible, and use a-priori information and information collected at run-time to best configure the system. Together with watchdogs, DENS corresponds to the Communication Infrastructure concern, and is the main contribution of the PhD thesis written by Katrine Stemland Skjelsvik [52].

Resource Manager, the component for Resource Management, aims at enabling best possible resource sharing among the devices involved in the network. During a rescue operation the involved personnel has a very strong incentive to collaborate and cooperate across organizations. This requires them to share knowledge and resources in order to fulfill their tasks. In a resource constraint environment such as a rescue operation, a distributed application needs the help of a resource manager in order to make the best out of the available resources. A resource manager's main duties in such environments are to register, discover services and data sources, and make the information available through the network. For this, each node can maintain a sharing profile with information about locally available resources and running services. The physical resources need to be frequently monitored, which can be achieved by using mechanisms provided by the operating system. Resource availability information can be disseminated in the network by using a shared data space as the one provided by the Data Management. Other alternatives are to announce availability of resources as notifications by using DENS, or to discover resources by querying the other nodes. Design of a non-intrusive and location information independent Resource Manager, deployed in a network with constrained devices and unreliable channels, is the main contribution of Ovidiu Valentin Drugan's PhD thesis [6]. The very part about Data Management, however, is being addressed in detail in this thesis.

Finally, the *Security and Privacy Manager* is in charge of Security Management. This component has a direct impact on the functionality of all the other components and therefore has to be considered from an early stage of development. The Security Management has to make sure that all the security requirements are fulfilled during the other components' operation. In addition to being something every other component depends on, the Security Management itself depends on some of the other components, such as key distribution, storage of keys and certificates, getting information on the neighborhood, etc. This cross-dependence between components, however, poses additional security issues that need to be taken care of. Security and privacy issues are being addressed in detail in this thesis.

2.2.2 MIDAS

MIDAS (*Middleware Platform for Developing and Deploying Advanced Mobile Services*, [9]) is a project funded by the European Commission (2006-2008). The main objective of the project is to simplify and speed up the task of developing and deploying mobile applications and services for larger heterogeneous networks. The two scenarios addressed by the MIDAS project include (but are not limited to) big sports events, e.g. Tour de France, and emergency and rescue operations. Although very different in their basics, these scenarios do share some common requirements and assumptions, such as: large number of users, limited duration of the event, networks consisting of a variety of different devices (possibly supporting different network technologies), the fact that infrastructure might be present but should not be solely relied on, strict consistency not being a requirement (i.e., if the situation does not allow it, it is better to get some information than no information at all). Next, we present shortly the three key features of MIDAS.

In heterogeneous networks, such as MIDAS is targeting, different devices might have *support for different of network technologies*, some of them might even support more than one. In general, this would cause compatibility problems. MIDAS, however, uses this fact to its advantage, by selecting the most adequate technology for communication between a group of nodes and thus extending the communication possibilities within the network, or even across different networks. This fact is transparent to the applications, which only need to know which nodes are available at any time, or possibly the network topology as well. If a node is not available at a certain point in time, an application may allow MIDAS to effectuate the message delivery at a later point in time when the recipient node is reachable again, by means of a store-and-forward mechanism.

To facilitate application development, MIDAS provides a higher level data abstraction known as *context information*, where context is interpreted as any information that could be relevant to the interaction between the user and the application. Context information is structured according to a domain model ontology which describes the current domain of deployment of the MIDAS middleware (e.g., the Tour de France race). MIDAS offers two context related services: *context operators* and *context-addressable messaging*, which rely very closely on the domain model structure. *Context operators* work as means for extraction of context information, possibly with synthesis of the information carried out by dedicated scripts (as an example, the application could query the middleware for the positions of users within a 500 m radius of another user) [5]. *Context-addressable messaging* is a best-effort service used to push messages to users described through their context (as an example, a message could be sent to all "Norwegian cyclists") [27]. By using an ontology based context model, both of these services can rely on underlying reasoning mechanisms and the information level interoperability between independent applications.

Information sharing by means of a distributed data space is addressed in the project by the MIDAS Data Space (MDS) middleware component. This component provides its users with a means of accessing the distributed shared data space without actually knowing where data are located. In addition, it performs transparent versioning of data, to allow for consistency management and conflict resolution in a dynamic environment, such as the one MIDAS is targeting. The MDS component represents the main focus of this thesis and will thus be described in more detail in the Chapter 4.

Chapter 3

Application Requirements

Emergency and rescue operations have specific characteristics and requirements. In the Ad-Hoc InfoWare [39] and MIDAS [9] projects, we have designed middleware services to facilitate the development of applications and services for emergency and rescue operations.

The content of this chapter is based on our previous work documented in form of a technical report [48]. The chapter includes three different example scenarios, followed by a requirements analysis of the scenarios' main aspects of interest for this thesis.

3.1 Emergency and Rescue Operations

Emergency and rescue operations are characterized by a number of organizations and individuals involved. As such, they form a heterogeneous composition with regards to organizational structure and devices present at the scene. However, by having a common goal, i.e., saving lives and limiting material damage, there must be mechanisms present that will cope with such heterogeneity and allow for efficient data sharing between the participants.

Organizations that are typically involved in such operations include the police department, paramedics, firefighters, armed forces, and possibly several additional organizations. In Norway, the police department has the main responsibility for rescue site management and coordination between all the organizations involved, as well as the county governor and other governmental departments who might get involved during the event. Even though each organization has its own set of rescue operation procedures and guidelines, cross-organizational interaction procedures and coordination structure must be defined. The coordination structure should to some extent shape the flow of information on the scene, even though the latter might sometimes deviate during the course of the event, if that improves efficiency.

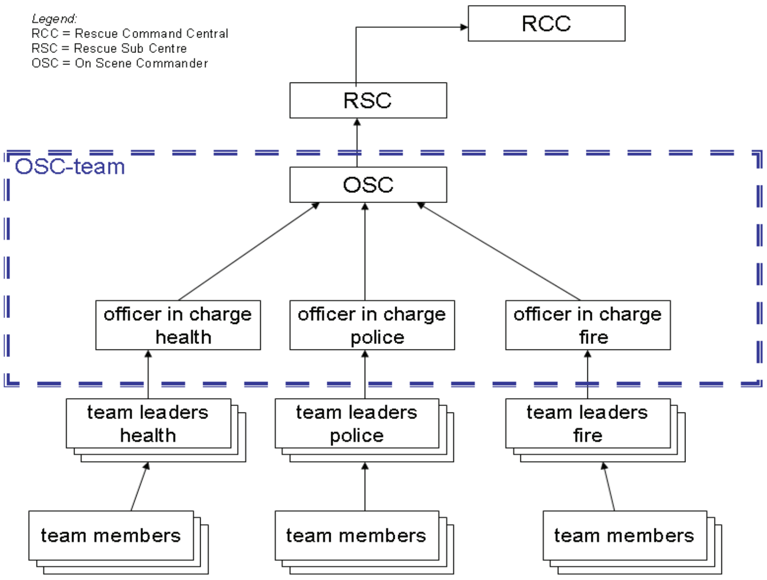


Figure 3.1: Organization and structure in rescue operations

In Norway, land operations are usually handled by the Rescue Sub-Centre (RSC), which has regional responsibility and appoints the on-scene coordinator/commander (OSC) at operation initiation. For larger operations, there is usually one person from each of the three main organizations (police, firefighters, and paramedics), in charge for the different aspects of the operation (public order, fire control, and medical treatment respectively), who all report directly to the OSC. On top of the hierarchy there is the Joint Rescue Coordination Command Central (RCC), whose role is mainly to monitor the operation and give advice. Figure 3.1 illustrates the organizational structure of rescue operations in Norway, as well as the role hierarchy and lines of reporting. The information is based on descriptions taken from the Norwegian Rescue and Search Service (SAR) [56]. It is this very model that we have used as basis for designing a key exchange protocol, presented and described in Chapter 8 (i.e., Paper #2).

3.1.1 Example Scenario: Earthquake

In this section, we present a scenario of a rescue operation after an earthquake. Even though it is a hypothetical scenario, it is based on information from a real earthquake that took place in western Nevada in September 1994 [7]. A strong earthquake may cause a high number of casualties and injured people, as well as severe material damages. This includes possible loss of electricity, communication infrastructure, blocked roads, etc., all of which might severely affect the rescue operation.

The rescue operation following such an earthquake may last for days or weeks, and will generally cover a large geographical area. Not only might blocked roads prevent rescue personnel's vehicles from coming to certain places, it might also hinder fuel supply to the same vehicles. Communication infrastructure might also be unavailable, either because cables have been damaged, or due to overload as a consequence of many people placing calls simultaneously. As a consequence, information sharing between the rescue personnel, necessary for a successful operation, might be highly challenging.

Efficient information sharing and means for information dissemination can speed up the work, which in turn may save lives. Such information include, but is not limited to patient files, drawings of buildings, maps, either from the Internet (if accessible) or from one of the devices present at the rescue scene (i.e., by forming an ad-hoc network). The fact that there are many different types of information causes a number of different flows of information, i.e., instructions from higher ranked officers to the next level in the hierarchy, status reports from the "ground people", information from sensors to people on the scene, information from experts not directly involved in the operation, etc.

3.1.2 Example Scenario: Railway Accident

A railway accident might happen in inaccessible terrain due to landslides, technical failure, sabotage, collisions, etc. The hypothetical rescue operation presented in this section is based on a serious train accident that happened in Norway in January 2000 [45].

The accident in question happens in a tunnel at a mountain pass, as a consequence of a rockslide outside the tunnel causing broken rail tracks. An incoming train hits the rocks on the way out of the tunnel, causing the locomotive to derail. There are a number of injured people, some of them still trapped in the train, while others managed to walk out. Both the locomotive and one of the train carriages are completely crashed, and there is high risk of fire. The train driver follows the procedure and reports the incident and location to the train control centre, who immediately forwards this information to the fire department and ambulance, and starts necessary emergency procedures. The temperature is -10°C, with deep snow in the area. The area has weak infrastructure and a need for special services. There is a mountain lodge located nearby, which can be accessed by a mountain road, and used for collecting evacuated train passengers. All personnel get relevant information (maps of the area, weather condition, available personnel and equipment, etc.) to their devices before leaving for the accident. RCC together with RSC starts a rescue operation to evacuate those in need of acute medical treatment which, due to harsh weather conditions, includes not injured people as well.

The tunnel, rocks, train carriages, and an area that is not very accessible – all of these hinder communication between the rescue personnel, and the rescue operation as a whole. Inside the tunnel, there is limited communication range. Outside, there is problem of communicating with both people inside, and people on the other side of the tunnel.

The leader of the first team arriving at the scene takes immediately the role of on-site commander (OSC), which he keeps until a higher ranked police officer arrives and takes over the role. The OSC gathers all necessary information, and coordinates equipment and personnel as they keep arriving. It is imperative to evacuate people from the carriages, something firefighters are in charge of. Medical personnel take over as people are outside of the tunnel, and categorize them by the degree of injury and need for acute treatment. They are then transported to the mountain lodge for further treatment.

Examples of communication flows in this scenario include information exchange among team members from the same organization (e.g., between medical personnel sharing registration and medical information about patients, or firefighters sharing temperature information in the monitored area, etc.), among *task forces* created on the spot between team members from same or different organizations (e.g., a team consisting of a few firefighters and paramedics, assigned to go through a certain train carriage to report on the situation), or communication between different levels in the rescue operation organizational hierarchy, e.g., RSC and OSC, team members and team leaders, or team leaders and OSC.

The landscape of the accident area has a big impact on the network topology, causing temporary or permanent partitions within or on the different sides of the tunnel. It must be ensured that all necessary data services are present in all the possible network partitions, and the mobility of the nodes has to be used as an advantage to deliver information across the partitions.

3.1.3 Example Scenario: Subway Station Accident

The following hypothetical scenario is located in an underground station of the urban metro system of Paris, France, based on a scenario description provided by the Régie Autonome des Transports Parisiens (RATP) [19].

The area has a good and well-maintained infrastructure, as opposed to the two previous scenarios. Still, some communication services might be unavailable at certain places of the subway system (e.g., tunnels between stations).

In this scenario, there are two trains standing on the opposite platforms of the same stations. One of the drivers notices smoke coming from one of his carriages, and immediately reports all the necessary details to the operation control centre (PCC). The PCC turns on the fire alarm, shuts down the traction power for the trains in the area of the station, asks the passengers to evacuate the station, and notifies the nearby stations. The information is then passed on higher up in the hierarchy, to the operation duty inspector (IPEX). The drivers of the two trains take pictures with their mobile devices and send them to the PCC, together with all necessary metadata (such as date, time, location, etc.). The IPEX calls for external support from the fire department, ambulance, police, etc., and informs the necessary instances about the accident. The passengers are evacuated and the station is secured.

The metro has a set of different emergency procedures, and a very precise hierarchy for the information flow, depending on who is on duty. All this information is readily available, which shortens the briefing phase. The incident manager, i.e., the first operation supervisor arriving at the scene, receives information from the mobile agents at the site and sends reports to the control centre. The IPEX keeps agents at nearby stations informed about the status of the accident. They may also send personalized to-do-lists to the agents, in form of tasks. When a task is completed, the information about this is immediately sent to the control unit.

Also in this scenario different organizations and groups of people are involved, e.g., train drivers, people working on the station, emergency teams, accident leaders form the metro department, etc. They all carry mobile devices for purpose of communication sharing, using infrastructure (e.g., GPRS or IEEE 802.11) where available, or forming an ad-hoc network otherwise.

3.1.4 Summary

The scenarios presented in the previous three sections are very different with respect to location, landscape, size of the accident area, number of people involved, available resources, time span, etc. While the earthquake scenario covers a large area and number of people, and might last for several days, the other two scenarios are more limited with respect to these factors.

On the other hand, the scenarios do share a lot of similarities, something that has to be looked into when designing a generic middleware. Many of the same organizations are involved in the scenarios (e.g., paramedics, firefighters, and police), with similar tasks, but with different numbers of people involved.

Possible sources of information include, for all scenarios, mobile devices carried by the rescue personnel, stationary devices, PCs in vehicles, sensors, possible access to data from the Internet, etc. Some of the information can be shared, but other information might be confidential (e.g., medical records). Infrastructure might be present, but should not be solely counted on due to possible breakages. Various types of information flow might be present in all the scenarios, e.g., among team members from the same organization, between different organizations, or between different levels in the rescue operation organizational hierarchy.

As part of the Ad-Hoc InfoWare project, we have identified six different phases [31] of a typical emergency and rescue operation.

Phase 1 – A priori

This phase is before any accident takes place, when the relevant organizations, in cooperation with the authorities, exchange information on data formats and shared vocabularies, and make agreements on procedures and working methods. Required certificates would be installed in this phase, and applications can be installed and run so as to allow completion of an initial self-configuration phase. A communication and knowledge environment tailored to relevant applications can be prepared, and data replication strategies chosen by the middleware. In a context aware system, contexts reflecting different scenarios can be prepared, group memberships based on user profiles set up.

Phase 2 – Briefing
This phase starts once the incident has been reported. The briefing involves gathering of information about the accident, e.g., weather, location, number of people involved, and facilities in the area. Some preliminary decisions about rescue procedures and working methods are also made at this stage. Based on information gathered during this phase, applications can be configured further, security levels chosen, and, if applicable, relevant rescue contexts and profiles put in force.

Phase 3 – Bootstrapping the network
This phase takes place at the rescue site, and involves devices joining and registering as nodes in the network on arrival. In addition, the appointing of rescue leaders takes place in this phase. By preparing communication and taking care of security restrictions in force, the middleware can improve the working environment of the applications.

Phase 4 – Running of the network
This is the main phase during the rescue operation, and the one this thesis focuses mostly on. Events that may affect the middleware services include nodes joining and leaving the network and network partitions and merges. Information is collected, exchanged and distributed. There may be changes in the roles different personnel have in the rescue operation, e.g., change of rescue site leader. New organizations and personnel may arrive and leave the rescue site, new groups of an ad-hoc, task-oriented kind may form, possibly involving people from different organizations. Applications communicate about available resources and capabilities of the nodes in the network, using whatever knowledge is provided by the middleware. It can update to changes in available resources as the network is evolving, query for more data or information as it becomes available, and adjust its configuration and behavior accordingly. Computing resources, processing environments and applications situated at neighbors can be utilized, using resource information provided by the middleware and obeying accepted policies for resource sharing. Replicas and proxies can be placed at strategic nodes in the network, and nodes can receive event notifications based on relevance and priority. As nodes join and leave the network the middleware can keep track of available resources and adjust its communication and knowledge environment accordingly. This is especially important for the location of database replicas, to ensure high data availability at all times.

Phase 5 – Closing of the network
At the end of the rescue operation all services must be terminated. Applications can adapt to the closing of the network by acting on received information about degradation of the capabilities and resources of the network.

Phase 6 – Post processing
After the rescue operation, operation specific data, e.g., resource use, user movements, and how and what type of information was shared, may be analyzed to gain knowledge for future situations. Depending on the nature of the application, it may have gathered statistical or other information for post scenario analysis or for future use.

Infrastructured networks cannot be relied on during the rescue operation itself. However, in the opening phases (phases 1-2), there are no such restrictions, which gives possibilities for preparations that to some degree can compensate for a lack of resources during the rescue operation.

3.2 Requirements Analysis

The three main aspects from the application scenarios of interest for this thesis are security (both towards and within the network), network characteristics, and data sharing.

3.2.1 Security Aspects

In emergency and rescue operations, where people's lives are at stake, certain aspects of security are of outmost importance. *Authentication* of rescue personnel and their devices, as well as data *integrity*, must be ensured in order to prevent intruders from injecting false information, which might allow them to disrupt communication and jeopardize the whole operation, or to gain access to *confidential* information. Certain types of data, e.g., medical records, are regarded as being strictly confidential. Not only intruders must be prevented from accessing them, but their access should be limited among the rescue personnel as well. For that reason, there must exist mechanisms that would ensure that only authorized persons/devices, groups, ranks, etc., have access to certain data.

The abovementioned issues are especially hard to achieve in an environment where wireless technologies are used for information exchange. Unlike wired networks, where one needs physical access to gain access to the network, the wireless medium allows anyone in the vicinity to be part of it, both passively and actively. It is fairly easy for an intruder to bring down the whole network by means of signal jamming, something that would need to be taken care of by physically locating and removing them. However, there are more subtle ways of achieving the same effect, i.e., by means of eavesdropping, injecting false information, or re-sending old but perfectly valid packets, that might be harder to locate and resolve. Another issue that is very much plausible, yet hard to be taken care of, is introduced by the fact that rescue operations are hectic and dynamic in nature, which makes it easy for a member of the rescue personnel to lose their (already authenticated) devices. If an intruder gets hold of such a device, and nothing is done to exclude it from the network, they might easily get the possibility to do anything the device's owner could do, something that in wrong hands could be disastrous.

All this has to be kept in mind when implementing a system that should meet the security demands for such a specific scenario as emergency and rescue operations.

3.2.2 Network Characteristics

Using MANETs on the rescue site is a promising approach, due to the fact that existing infrastructure might not be present or might be destroyed. However, the layout of the affected area, physical obstacles, and the mobility of the rescue personnel and their devices, might lead to frequent and/or long term network partitions.

In the train accident described in Section 3.1.2, rescue personnel needs to take care of passengers at both sides of the tunnel, causing two network partitions. There might be ways to connect these partitions (e.g., by means of GSM, satellites, or ad-hoc placed base stations above the tunnel if possible), but they need to be able to function properly even if this is not possible. In addition, some teams might go into the tunnel, causing new partitions that also need to function independently.

The rescue personnel's devices might be very heterogeneous with respect to hardware (PDAs, mobile phones, laptops, stationary computers within vehicles, etc.) and available resources (e.g., battery power, storage space, CPU, etc.). Both the heterogeneity of devices, and the broad range of functional and non-functional requirements, impose the need for resource management mechanisms. In order to have efficient resource management mechanisms, prediction of network connectivity is an important, albeit challenging factor.

Node mobility in a MANET poses challenges also with regards to packet routing, creating a need for specialized routing protocols, to be able to find paths between pairs of nodes. Given the fact that there is no infrastructure in a MANET, each node needs to be able to act both as an end-user device and a router (as seen from a traditional network's perspective). This issue is described in more detail in Section 2.1.1.

3.2.3 Data Sharing

Due to the dynamicity of the network, data sharing poses challenges with regards to storing, locating, and retrieving data. The fact that a certain node might be out of reach could cause important data to be unavailable. To ensure availability of data, even if the network is partitioned, the data must be replicated on more than one node, and consistency between replicas needs to be taken care of. A system such as MIDAS Data Space (see Chapter 4) can be used to address this issue.

The fact that a node might not be reachable at a certain point of time makes it also necessary to develop protocols for a more asynchronous, i.e., *delay tolerant* communication. This can be achieved e.g. by means of a publish/subscribe mechanism, in combination with mechanisms such as store-carry-forward, epidemic routing [59], message ferrying [63], etc., to deliver notifications. Data sharing in emergency and rescue operations is not a trivial issue, not only due to the fact that the network is dynamic, but also because each organization might use its own standardized vocabulary and data model. Therefore, it is desirable to support subscriptions that are targeting different vocabularies, including translation or mapping between different vocabularies.

Chapter 4

MIDAS Data Space

The core purpose of the MIDAS Data Space (MDS) is to provide applications and other MIDAS components (so called *MDS users*) running on different nodes a mechanism to share information by inserting data in and retrieving data from a shared data space. To the MDS users, the shared data space is very similar to a standard relational database, with functions to create, add data to, retrieve data from and remove data from tables. These MDS functions work in such a way that data added by an application on one node is made available for retrieval on other nodes, without any intervention needed from the application program. The shared data space is implemented using a combination of data replication and remote operations, but this fact is transparent to applications. To the applications, it looks as if there is just one instance of each table, accessible from all nodes. However, multiple table instances (replicas) may exist in the network, providing higher reliability in case of network partitioning. MDS uses a cross-layer approach when performing decisions on where replicas should be placed, taking into account both the network topology and applications' access pattern to the data. To ensure eventual consistency, conflict resolution and auditing, MDS implements data versioning, i.e., data are never physically deleted and, if updated, old versions are kept.

In short, MDS can be summarized by the following main features:

- shared data space where data location and distribution issues are transparent to the users
- two types of replication (*eager* and *lazy*), to ensure data availability and eventual consistency between the replicas
- data versioning, for purposes of consistency, conflict resolution, and auditing
- cross-layer optimizations, to minimize the resource usage due to data replication

In this chapter, we describe in detail the MDS component, as well as the proof-of-concept implementation used as the basis for this thesis. Due to time constraints imposed by the project's duration and deadlines, the proof-of-concept implementation of the MDS, as presented in this chapter, has certain limitations. The focus was put on getting the component functional, and as a consequence not all subcomponent implementations take into account issues such as scalability and resource usage. These issues were further studied after the project's end, and are presented in Chapter 13.

4.1 MDS within MIDAS

MDS is one of the core components of the MIDAS Architecture, as shown in Figure 4.1, taken from [53]. MDS uses mostly services from the Connectivity and Routing component, as well as some Common Functions. MDS offers its services to both applications and other MIDAS components, mostly the Context Space component.

4.2 MDS Architecture

The MDS component consists of the following sub-components:

- *Query Analyzer (QA)* – supports queries towards the local database, as well as towards remote nodes.
- *Global MetaData Manager (GMDM)* – keeps track of which tables have replicas on which nodes.
- *Data Allocator (DA)* – decides where in the network replicas of each table should be placed.
- *Data Synchronizer (DS)* – performs synchronization of data between nodes having replicas of the same table.
- *Subscription Manager (SM)* – offers publish/subscribe functionality where an application may subscribe to changes to a table and receive notifications when and if such a change takes place.
- *Local Storage (LS)* – manages data persistence on behalf of the other MDS subcomponents.

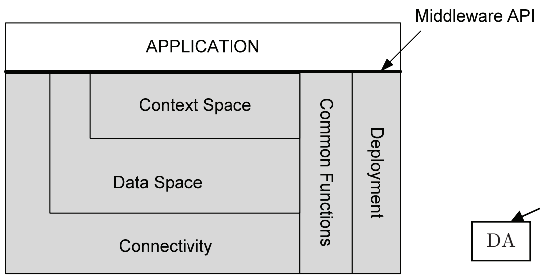


Figure 4.1: MIDAS Architecture

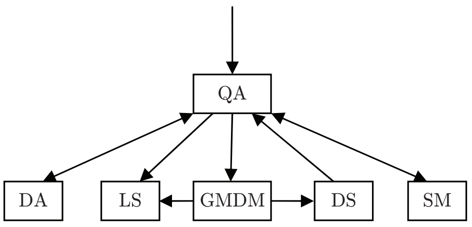


Figure 4.2: MDS Architecture

Figure 4.2 shows the overall design of MDS and the interaction between its sub-components. Here are some observations:

- QA is the only sub-component that other MIDAS components (or applications, through the Middleware Core) have contact with, serving as a so called *façade* component.
- The only sub-components communicating directly with the database (i.e., LS) are QA and GMDM.

The QA sub-component receives queries containing INSERT, SELECT, UPDATE or DELETE statements. It is the task of MDS to choose on which node a query will be performed, and thus provide applications with transparency on where data are located. In addition, MDS users are given the possibility to make that decision themselves in case they might have knowledge about where a table is (or should be) stored. Being a façade component towards MDS, QA also implements the functionality for allocating or de-allocating table replicas, as well as informing GMDM on such events, although the decision on where to place replicas is the realm of the DA sub-component. QA performs transparent versioning of data, as explained in Section 4.4 of this thesis. QA takes care of immediately propagating data changes that might take place as a consequence of a query, to all replicas in the network. This process is called *eager synchronization*.

The GMDM sub-component keeps track of which table replicas are located on which nodes. Ideally, all the nodes' GMDMs should have the same view of the network. For this purpose, when a new table is created, GMDM stores new metadata information on the table in question and immediately starts a metadata synchronization process with its neighbors. When a query is received from the application, it is GMDM who knows where the query can be performed. When a new node joins the network, the GMDMs synchronize metadata information, thus getting an updated view on what table replicas exist on which nodes. When done, GMDM calls DS to synchronize the actual data in tables that the local node has in common with the newly arrived node. This process is called *lazy synchronization*.

The DA sub-component decides when and where in the network instances of each table (i.e., replicas) should be placed. As part of this thesis, we have implemented a solution where an instance of DA on each node continuously monitors usage patterns (local and remote) of each table. This information is periodically sent to a central node, called Traffic Controller (TC), whose DA sub-component uses it to actually decide on the placement of each replica. Work on the DA sub-component, including evaluation results, has been described in detail in Chapter 13 (i.e., Paper #7).

The LS sub-component gives access to local data, stored in a relational database. LS makes it possible for any node to choose any of the supported relational database management systems (see Section 4.3 for details).

The DS sub-component is used to perform *lazy synchronization* of data between nodes having replicas of the same table. When a change in the network is reported by the MIDAS Communication and Routing component, involving one or more nodes that have entered the network, a synchronization process is started.

After the GMDMs have synchronized their metadata information, they each call their respective DS to start peer-to-peer synchronization on the data level with nodes having replicas of the same table. Due to network disruptions and messages being lost for other reasons, the abovementioned procedure might not be sufficient. To make sure that eventual consistency is obtained, DS therefore performs periodic synchronization with nodes with which it has tables in common.

The SM sub-component provides a subscription and notification mechanism. MDS users may subscribe to changes to certain fields in a certain table (e.g., "*notify me if Paul's temperature gets over 38°C*").

4.3 Database Support

In order to be able to run the LS sub-component of MDS, a 3rd party DBMS solution is used, with the responsibility for managing table definitions and their information content locally at a node. The functionality provided to MDS users through the MDS information management API will be a subset of what the DBMS solution provides. The exact subset of SQL calls supported by MDS is described in Section 4.5.

MDS currently supports the following free SQL databases:

- *H2* [18] is implemented in Java and may run within the middleware or as a standalone server. The database is stored on disk.
- *Sqlite* [51] is implemented in Java, and runs within the middleware. It can be chosen whether the database during operation will only be kept in memory or on disk.
- *HSQLDB* [17] is implemented in Java, and runs within the middleware. It can be chosen whether the database during operation will only be kept in memory or on disk.
- *MySQL* [33] is not implemented in Java and therefore must be run separately. The database is stored on disk.

Since synchronization of tables is done by means of standard SQL statements, different nodes may run different types of databases (e.g., one node may use H2, while another one uses HSQLDB), not affecting synchronization of shared tables.

4.4 Data Versioning

Contrary to standard relational databases, MDS provides transparent versioning of data. Instead of doing updates and deletions on the data items, the items are marked as obsolete and a new prevailing (valid) version is created. For this purpose, each table has a *Status* field, which can have one of the following values:

- *VALID* – denotes a valid record, i.e., it will be included into the result set when an application sends a query, and it will be considered for a user's UPDATE or DELETE statement.

- *UPDATED* – denotes a record that once was valid, but has been updated since. Such records are kept for book-keeping and auditioning purposes and, unless explicitly asked for, they will *not* be included into the result set of a query, and neither will they be considered for a user's UPDATE or DELETE statement.
- *DELETED* – denotes a record that has been deleted. Such records are kept for book-keeping and auditing purposes only and will never be considered for any user request.

All three types of records are used by MDS to perform limited automatic conflict resolution during the lazy synchronization process. This process is illustrated by examples in Section 4.8.

The primary key in a table is a system generated number (incremented for each new record), called *Local_ID*. The *Local_ID* field is only used by the local database on each node and is not included in the synchronization process. For synchronization purposes, we have an additional identifier (called *ID*), consisting of the fields *NodeID_Created* and *TS_Created*. The *NodeID_Created* and *TS_Created* fields are set when a new record is inserted into a table, and contain the node ID and timestamp respectively, and are later used by MDS to identify whether two records are actually different versions of the same record.

Since versioning is performed internally by MDS, applications cannot define primary keys as they would in a traditional relational database. The reason for this is that, due to versioning, there may exist more than one record having the same value for such a key. It is however possible to define a so called *application defined key*. Each table can have one field that will act as such a key. By doing this, the application developers tell MDS upfront what they consider to be an identifier, and MDS later uses that field as the basis for doing conflict resolution.

4.5 Data Storage and Retrieval

MDS allows storage and retrieval of data by means of SQL statements. When a query is received by MDS, it is the QA component that parses it. After determining the type of SQL statement and the table or tables being affected, QA needs to know on which node the query should be run. This procedure can be explained by the following pseudo code:

```
if destination node specified {
     perform query on specified node
} else {
     ask GMDM which nodes have replicas of all the tables in question
     if local node on the list
          send request to the local LS component
     else
          send request to a remote node from the list
}
```

In the current implementation, GMDM first sorts the nodes according the rules for greylisting and blacklisting, explained later in Section 4.11. Next, the nodes are sorted by the number of hops. Finally, QA selects the first node from the list if it is a remote query.

The next step is to process the query. Here, we describe in detail what happens for each of the four types of queries supported.

SELECT statement

Syntax:
```
SELECT <fields> FROM <table>
[WHERE <condition>]
[GROUP BY <fields>]
[HAVING <condition>]
[ORDER BY <fields> [ASC | DESC]]
[LIMIT <number>]
```

A SELECT statement is always performed only on one node. It returns the result of the query as a result set if it succeeds, or a null-pointer if it fails on local queries. On remote queries, a timeout exception is thrown if the remote node does not respond within the predefined timeout period. The default period is set to 30s, unless it is overridden in the configuration file, or within the query call itself.

It is possible to specify more than one table on which to perform the query (so called joined query), however a joined query will only work if there exists a node hosting replicas of *all* the tables in question. To be able to use this functionality and rely on it, applications should have knowledge on where data are stored. This is, for instance, possible if the applications want to force creation of a table replica on a certain node. Even though this might contradict with the principle of transparency, it gives applications useful additional functionality that otherwise would not be available.

Table aliasing in the SELECT statement is not supported.

With respect to versioning, it is possible to specify which and what kind of records that should be returned. Some examples are presented in Section 4.8.2.

INSERT statement

Syntax:
```
INSERT INTO <table> (<fields>) VALUES (<values>)
```

An INSERT statement adds a record to the table given in the SQL statement, and if an instance of this table does not exist yet, it is created on a node picked by DA. If it is a table with an existing instance, and there are replicas within the network, the record is immediately sent to all the replicas on the nodes that are reachable (i.e., *eager synchronization* is invoked).

If the INSERT statement caused creation of a new table replica, QA notifies GMDM on this fact, which then makes sure metadata are synchronized on all nodes. If replicas of the given table already existed prior to the creation, which could happen if this latest creation was a result of an application forcing the operation to be performed on a specific node, lazy synchronization takes place to make sure the newly created instance has a copy of the old data as well.

The status of the newly inserted record is automatically marked as *VALID*.

UPDATE and DELETE statements

Syntax:
```
UPDATE <table> SET <fields_and_values>
[WHERE <condition>]
DELETE FROM <table>
[WHERE <condition>]
```

From the application point of view, an UPDATE statement updates and a DELETE statement deletes a record or records, just like any other relational database. As with INSERT, any changes will be immediately propagated to all other reachable nodes having replicas of the same table. If a remote node or nodes are not reachable, the eager synchronization operation will not be carried out, and lazy synchronization will take care of it when these nodes become reachable.

In reality, UPDATE and DELETE statements are transformed by MDS to perform versioning of data. An UPDATE statement first marks the old record(s) matching the given condition as *UPDATED*, after which *VALID* version(s), with the new values, are inserted. A DELETE statement marks the old record(s) matching the given condition as *DELETED*.

4.6 Data Replication

Multiple instances (replicas) of each table might be present in the network. Contrary to some other systems, MDS does not make a distinction between a main (often called *master*) copy and secondary ones, i.e., all replicas are treated equally.

The proof-of-concept implementation of MDS includes a simple solution for doing replication and synchronization, and only a few performance issues are taken into consideration. When the DA decides where to create a table, it always picks the local node. For performance and reliability reasons, the application can request MDS to create a replica of a specific table on a specific node. Information about this operation is sent to the local GMDM. If the node in question is not reachable at the given moment, the table will not be replicated. Even though such a solution was good enough for the MIDAS prototype, it clearly has room for improvements. Improving the functionality of the DA sub-component is one of the main contributions of this thesis, presented in detail in Chapter 13 (i.e., Paper #7).

Having multiple table replicas introduces the need for synchronization, to make sure that data are consistent as much as the situation allows it. MDS supports and implements two synchronization concepts, namely *eager* and *lazy* synchronization.

Eager synchronization takes place whenever a change is done in the data space (by means of an INSERT, UPDATE or DELETE statement). The change is immediately propagated to all replicas within reach, to ensure that the nodes in the network have a consistent view of the data. If a node or group of nodes were out of

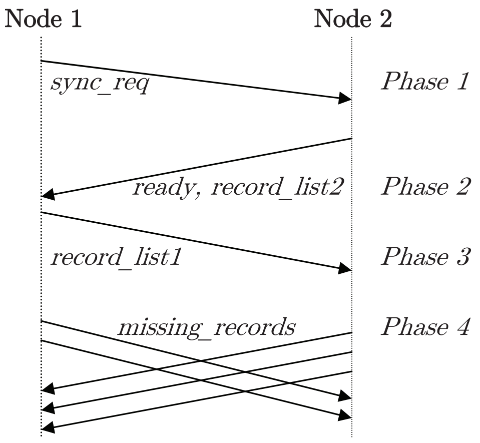


Figure 4.3: Lazy synchronization process

reach at the moment a change occurred, lazy synchronization takes care of possible inconsistencies that might be present after the two network partitions have merged.

Lazy synchronization of data is performed when CRT detects and reports a new node in the network. To be on the safe side, this is also performed periodically. The synchronization process between two nodes that have tables in common is performed by their respective DS sub-components. Synchronization is based on the records' key field *ID*, which uniquely identifies a record and all its versions. Inconsistencies are detected when records with the same key field have differences within other MDS fields (i.e., multiple versions of a record exist), thus introducing need for conflict resolution.

The synchronization process in the current implementation of MDS itself is simple and not scalable. It is divided into four phases, as shown in the message sequence diagram in Figure 4.3:

1. Node 1 sends a request for synchronization. In the case Node 2 already has an ongoing synchronization process and cannot respond, it will queue the request.
2. When Node 2 is ready for synchronization, it responds with a list of records it has for each table they have in common. Each record is identified by the field *id*. The field *TS_Changed* (the time when the record has been changed) is also sent for conflict resolution purposes.
3. Node 1 responds with an equivalent list.
4. Nodes 1 and 2 send each other records the other node does not have or has an old version of (see below for details); this is done by issuing remote queries using QA.

In case of inconsistencies within a table after a network merging, a simple conflict resolution strategy is performed, and timestamps are being looked into. Provided that two nodes had a record in common (i.e., the same version of the record) before partitioning, the following cases can occur when two nodes compare a pair of records upon merging:

1. One or both nodes updated the record in the meantime – only the newest *VALID* record is kept as such, while the others are marked as *UPDATED*. Each node sends over all the versions the other node does not have.
2. One node has deleted the record – the version with the newest timestamp "wins".
    a. If the deleted record has a newer timestamp, it will be deleted (i.e., marked as *DELETED*) on the other node as well.
    b. If the other node performed an update (and thus created a fresh *VALID* version) *after* the first node's delete, the record will become *VALID* again.

Even though the fact that a record is newer or older may not be correct due to unsynchronized clocks, it will be unambiguous. This is explained in detail in Section 4.11.

MDS does have knowledge of what an application considers to be a record key. If a record is inserted concerning the "same" data (as seen from the application's point of view) on two different nodes, this will result in two separate records concerning the "same" data. Therefore, the lazy synchronization process will *not* compare these records, and conflict resolution will be done either by the application or by MDS if requested. This has to be done each time a query is performed on such records.

By performing experiments in the emulation environment NEMAN (see Chapter 8), we were able to verify that the protocol indeed leads to consistent state where all nodes have the same view on the tables.

4.7 MDS Schema

The MDS Schema contains definitions of properties that MDS must relate to at startup, i.e., database table definitions. These definitions cannot be changed until MDS is terminated and restarted. The MDS Schema is described by one or more plain text files using a simplified XML. The default file is named *mds-schema.xml*, which must be present and unchanged. When MDS is started, all valid files are loaded into the memory to form one single schema. Therefore, it is important that tables have unique names and that there is only one definition for each table.

MDS supports two different types of tables: tables that only can be accessed locally at a node (*Local Table*) and tables containing information for sharing between nodes (*Shared Table*). Local and shared tables have identical structure; the only difference is that data in shared tables is made available to all nodes and that shared tables potentially have multiple replicas.

The following properties apply to both local and shared tables:

- *Name*: A unique name within the active MDS domain.
- *Scope*: Specification whether the table shall be a *Local Table* or a *Shared Table.*
- *Fields*: The names and types of all the fields that can hold data when the table is in use. Supported data types correspond to a subset of the types supported by the 3rd party DBMS to be used. Optionally, *one* field in each table can be set as an application defined key, used by MDS for conflict resolution.
- *Constraints*: Possible additional constraints (e.g. secondary keys).

There is a set of fields predefined for every table. The values in these fields are required for MDS internal "book-keeping" functionality and, in addition to that, they may be read by MDS users as any other field (but not modified). The following default fields are defined:

- *Local_ID:* The unique identifier of a certain row – this field is *not* shared with other nodes as it is automatically assigned by the database and might otherwise result in conflicts.
- *ID:* Shared unique identifier of a record or different versions of the same record, consisting of *NodeID_Created* and *TS_Created*. This can be used by applications and is not dependent on location of data. This serves as the main key used for identifying records that might be out of sync on various nodes.
- *NodeID_Created*: The ID of the node that created the record.
- *TS_Created*: Timestamp, generated on the node on which the INSERT operation was performed that resulted in creation of the record.
- *NodeID_Changed*: The ID of the node that last changed the record.
- *TS_Changed*: Timestamp, generated on the node on which the last UPDATE, DELETE or INSERT operation was performed that resulted in a change or creation of the record. This is used for conflict resolution if inconsistencies are detected.
- *Status*: Indicates the status of the records, possible values are: *VALID*, *UPDATED* or *DELETED* (see Chapter 4.4 for details).
- *Record_Replaced:* Used for book-keeping purposes, contains the *Local_ID* of the previous version of this record (or NULL, if this is the original version).

An example of the MDS Schema can be seen in Figure 4.4. The first line is present for XML compatibility (although it is optional) in case some other XML parsers are used rather than the simplified parser developed for this particular implementation of MDS. To reduce processing time and memory, this simplified parser does not take into account the content (value) within XML tags, i.e., everything needs to be specified as attributes.

The default scope of a table is *Shared*, unless specified otherwise. If it is specified as *Local*, no contact with GMDM is established when performing actions on the table. For that reason, there cannot exist both a shared- and a local-scope table with exactly the same name.

Optionally, *one* field in each table can be set to be an application defined key (as the field *description* in table *burek*, in the above example), which does not guarantee that there will not be more than one record with the same key. This key is only looked into when MDS performs conflict resolution in the case when multiple rows with the same key are returned as a result of a query using the parameter *VERSION_LATEST*. However, since the change on more than one version can happen at exactly the same time, applications should be able to handle receiving a response with more than one record. Due to versioning, it is not allowed to have additional database defined custom keys or unique-constraints.

```
<?xml version='1.0' encoding='iso-8859-1'?>
<mds>
  <table_definitions>
    <table name="burek">
      <field name="description" type="text" key="yes"/>
      <field name="price" type="int"/>
    </table>
    <table name="pita" scope="local">
      <field name="description" type="varchar(64)"/>
      <constraint/>
    </table>
    <default_fields>
      <field name="local_id"        type="int" constraints="auto_increment"/>
      <field name="id"              type="varchar(128)"/>
      <field name="nodeid_created"  type="varchar(64)"/>
      <field name="nodeid_changed"  type="varchar(64)"/>
      <field name="ts_created"      type="varchar(16)"/>
      <field name="ts_changed"      type="varchar(16)"/>
      <field name="record_status"   type="varchar(64)"/>
      <field name="record_replaced" type="int null"/>
      <constraint name="unique (nodeid_created, ts_created, ts_changed,
                        record_status)"/>
    </default_fields>
  </table_definitions>
  <type_conversion>
    <database type="h2">
      <convert from="auto_increment" to="auto_increment(1, 1) primary key"/>
    </database>
    <database type="sqlite">
      <convert from="auto_increment" to="primary key autoincrement"/>
      <convert from="varchar"        to="text"/>
      <convert from="int"            to="integer"/>
    </database>
    <database type="hsql">
      <convert from="auto_increment" to="identity primary key"/>
      <convert from="text"           to="varchar"/>
      <convert from="number"         to="int"/>
    </database>
    <database type="mysql">
      <convert from="auto_increment" to="primary key auto_increment"/>
    </database>
  </type_conversion>
</mds>
```

Figure 4.4: Example of the MDS Schema

```
CREATE TABLE burek (
  description text,
  price int,
  local_id int auto_increment(1, 1) primary key,
  id varchar(128),
  nodeid_created varchar(64),
  nodeid_changed varchar(64),
  ts_created varchar(16),
  ts_changed varchar(16),
  record_status varchar(64),
  record_replaced int references burek(local_id),
  unique (nodeid_created, ts_created, ts_changed, record_status)
)
```

Figure 4.5: Automatically generated SQL statement for creating tables

The *default_fields* are read from *mds-schema.*xml and are added to each table at the moment of its creation. The creation of the abovementioned table *burek* will result in the SQL statement towards the DBMS as shown in Figure 4.5.

All the tables have a *Local_ID* field as the primary key which is an integer and is automatically incremented. The described syntax for auto_increment is specific for H2 DBMS and might differ if another DBMS is used. For that reason, MDS has methods to adjust the syntax of CREATE TABLE to fit other types of databases (details on how this is done are not described here as it is out of the scope for this document). The other field types and constraints are standard.

The *type_conversion* section gives the possibility to define conversion rules for field types or other syntax that might be different in different databases, making it sure that a correct CREATE TABLE statement is generated. As an example (shown in Figure 4.4), the statement to create the table's primary key, and make it automatically increment for each newly inserted record, is different in each of the abovementioned databases.

4.8 Examples

In this section, we show some concrete examples of how MDS uses versioning to cope with network partitions, as well as an example of what would happen in such a situation if versioning was not present.

4.8.1 Example: Update While Network is Partitioned

The table *Patients* is allocated on *Node 1* and *Node 2*. At first the two nodes are in the same network partition. Then an application on Node 1 inserts data about patient John into the table. This is also stored on Node 2 by using eager synchronization. Both nodes have the same view of the table, as seen in Table 4.1.

Table 4.1: Patient table on nodes 1 and 2, before partitioning

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	1	100	Valid	John	80

Next, the two nodes end up being out of range of each other.

A new record about a new patient, Paul, is inserted into the table on Node 1. Node 1 also has an update about the patient John. This results in that record #1 changes the *Status* and *TS_Changed* fields accordingly, and there is an insert of a new record about John (see Table 4.2)

Table 4.2: Patient table on Node 1, during partitioning

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	~~1~~ 1	~~100~~ 102	~~Valid~~ Updated	John	80
2	1	101	1_101	1	101	Valid	Paul	90
3	1	100	1_100	1	102	Valid	John	90

On Node 2*,* there is an UPDATE about John at time 103, as well as an INSERT about patient Paul at time 104 (see Table 4.3).

Table 4.3: Patient table on Node 2, during partitioning

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	~~1~~ 2	~~100~~ 103	~~Valid~~ Updated	John	80
2	1	100	1_100	2	103	Valid	John	95
3	2	104	2_104	2	104	Valid	Paul	92

When the partitions merge at time 105, and these two nodes meet, lazy synchronization of all records is performed. The results can be seen in Table 4.4 and Table 4.5.

- Record #1 is considered to be the same.
- Record #3 at Node 1 and Record #2 on Node 2 have the same key (ID) and are both VALID. Node 1's record is older and is therefore marked as UPDATED. Then, both records are exchanged.
- Record #2 at Node 1 and Record #3 at Node 2 are from MDS' point of view not the same record, even though it is about the same patient, so each node sends an INSERT statement to the other node, and there are therefore two records about Paul that are valid. The result is the following:

Table 4.4: Patient table on Node 1, after the two partitions have merged

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	1	102	Updated	John	80
2	1	101	1_101	1	101	Valid	Paul	90
3	1	100	1_100	~~1~~ 1	~~102~~ 105	~~Valid~~ Updated	John	90
4	1	100	1_100	2	103	Valid	John	95
5	2	104	2_104	2	104	Valid	Paul	92

Table 4.5: Patient table on Node 2, after the two partitions have merged

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	1	103	Updated	John	80
2	1	100	1_100	2	103	Valid	John	95
3	2	104	2_104	2	104	Valid	Paul	92
4	1	100	1_100	1	105	Updated	John	90
5	1	101	1_101	1	101	Valid	Paul	90

4.8.2 Example: Query

Here we present a few examples of queries performed with different version control parameters. The values in the table are as shown in the previous example, and the applications on both nodes get the same results.

- Query: `SELECT id, name, pulse, status FROM Patient`
  Version Control: LATEST
  Result:
  ```
  1_100 John    95    Valid
  2_104 Paul    92    Valid
  ```

  Discussion:

  Only the *newest VALID records* (as denoted by the *TS_Changed* field) are returned for each application defined key (in this case *Name*). In the case of Paul's record, this means that the record with the ID *1_101* will not be returned since the record *2_104* has a newer timestamp.

- Query: `SELECT id, name, pulse, status FROM Patient`
  Version Control: VALID
  Result:
  ```
  1_100 John    95    Valid
  2_104 Paul    92    Valid
  1_101 Paul    90    Valid
  ```

  Discussion:

  All VALID records are returned, not only the newest ones. Thus, in the case of Paul's record, this time both *1_101* and *1_104* are returned.

- Query: `SELECT id, name, pulse, status FROM Patient`
  Version Control: ALL
  Result:
  ```
  1_100 John    80    Updated
  1_100 John    95    Valid
  2_104 Paul    92    Valid
  1_100 John    90    Updated
  1_101 Paul    90    Valid
  ```

Discussion:

All non-DELETED records are returned. The example above does not contain any deleted record, causing all five records to be returned. The application should check the *Status* field to make sure it handles the records correctly.

4.8.3 Example: Delete or Update While Network is Partitioned

Again, we have a stable network with a situation where both nodes have the same data, as seen in Table 4.6.

Table 4.6: Patient table on nodes 1 and 2, before partitioning

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	1	100	Valid	John	80
2	1	101	1_101	1	101	Valid	Paul	100

After partitioning, the following events occur:

- Node 1 deletes both the records (John and Paul) at time 105 (see Table 4.7)
- Node 2 updates the record John at time 106 (see Table 4.8)

Table 4.7: Patient table on Node 1, during partitioning

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	~~1~~ 1	~~100~~ 105	~~Valid~~ Deleted	John	80
2	1	101	1_101	~~1~~ 1	~~101~~ 105	~~Valid~~ Deleted	Paul	100

Table 4.8: Patient table on Node 2, during partitioning

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	~~1~~ 2	~~100~~ 106	~~Valid~~ Updated	John	80
2	1	101	1_101	1	101	Valid	Paul	100
3	1	100	1_100	2	106	Valid	John	81

Table 4.9 and Table 4.10 show the situation at time 110, after merging has occurred:

- Record #1 on Node 1 and Record #3 on Node 2 are compared. Node 2 made the change most recently, and the record is "reincarnated" on Node 1.
- Record #2 on Node 1 and Record #2 on Node 2 are compared. Node 1 made the change most recently, and the record on Node 2 is also marked as *DELETED*.

Table 4.9: Patient table on Node 1,after the two partitions have merged

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	1	~~100~~ ~~105~~ 110	~~Valid~~ ~~Deleted~~ Updated	John	80
2	1	101	1_101	1	~~101~~ 105	~~Valid~~ Deleted	Paul	100
3	1	100	1_100	2	106	Valid	John	81

Table 4.10: Patient table on Node 2, after the two partitions have merged

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	~~1~~ 2	~~100~~ 106	~~Valid~~ Updated	John	80
2	1	101	1_101	~~1~~ 2	~~101~~ 105	~~Valid~~ Deleted	Paul	100
3	1	100	1_100	2	106	Valid	John	81

4.8.4 Example: Problems When No Versioning

The following simple example shows the importance of the versioning mechanism in dynamic scenarios, such as the ones targeted by this thesis. If versioning were not available, deleting a record in a table replicated on nodes in different partitions would create a (clearly undesirable) situation where this record would be inserted again after merging of two partitions.

Initially, both nodes have the same view, as seen in Table 4.11.

Table 4.11: Patient table on nodes 1 and 2, before partitioning

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
1	1	100	1_100	1	100	n/a	John	80

While the network is being partitioned, the record is deleted at Node 1 (Table 4.12), and the table becomes empty.

Table 4.12: Patient table on Node 1, after the record has been deleted

Local_ID	NID_Created	TS_Created	ID	NID_Changed	TS_Changed	Status	Name	Pulse
~~1~~	~~1~~	~~100~~	~~1_100~~	~~1~~	~~100~~	~~Valid~~	~~John~~	~~80~~

Node 2 does not have knowledge of this and keeps the table unchanged.

After the two partitions merge, synchronization takes place and the record gets “reincarnated” at Node 1 (again, as seen in Table 4.11), because no mechanism is in place that would make Node 1 remember it already had deleted that very record.

4.9 Subscription and Notification Mechanism

MDS' sub-component SM provides applications with a subscription and notification mechanism. Subscriptions are stored in a shared table, which means that every node knows about every other node's subscriptions.

The main subscribe-method's parameters are the *table name* and *rules*. For a subscription to be triggered, all the given rules must be satisfied. The operators supported are: "<", ">", "=", "<=", ">=" and the *string contains* operator, denoted by "~". The subscriber application must implement a call-back method that is called when a notification is received. In the current implementation, subscriptions do not have specified time validity, and must therefore be unsubscribed from when there is no need for them any more.

Since the subscriptions table is updated and synchronized by MDS, it is not needed to send subscriptions and un-subscriptions to specific nodes having specific tables, neither it is necessary to keep track of new nodes getting new replicas. This solution is however not very scalable.

Having a fully replicated subscriptions table may result in multiple notifications as a notification is going to be sent from each node where a change of a table that satisfies a subscription is stored, thus increasing the probability that the application will receive the notification. The SM component at the subscriber's node makes sure that the subscriber only receives the notification once.

4.10 Performance Experiments on the MDS Prototype

Detailed performance experiments of the MDS prototype are presented later in this thesis, in Chapters 12 and 13 (Papers #6 and #7). A number of experiments, however, were omitted in those papers due to space limitation. In this chapter, we present some of these experiments.

4.10.1 Memory Cards in Nokia Internet Tablets

This set of experiments was performed to see whether the physical location of the database on the device is important, with regards to access times. The experiments were performed on three generations of the Nokia Internet tablet available at the time (770, N800 and N810), which were candidates for target devices in the MIDAS project.

Table 4.13 shows the results for the sqlite database on each of the three devices. The labels I, S, U, and D represent the INSERT, SELECT, UPDATE and DELETE statements respectively. The numbers represent access time (in milliseconds) per record when an SQL statement is run that affects 100 records (except for INSERT, which had to be run 100 times individually). Empty fields mean that the given device/memory combination is not supported due to hardware limitations. Experiments on other supported databases (HSQLDB, H2, and MySQL) followed the same pattern and are therefore omitted. Sqlite and HSQLDB are the only databases that support keeping data in memory, instead of storing it on permanent storage. The reason why the UPDATE numbers are so high lies in the fact that each original UPDATE statement is, due to versioning, in reality executed as 1x SELECT, 1x UPDATE and *N*x INSERT statements, where *N* denotes the number of records affected by the original statement.

It can be seen that data location and type of memory (ram, internal or external) can play a difference with regards to access times. If the database is expected to be large, however, the only practical option remains using an external memory card. We tested several memory cards (depending on each device's support), and results between them did not indicate that the type or speed of external memory card would make a significant impact.

Table 4.13: Access times for the sqlite database on Nokia Internet tablets

Device	Nokia N810				Nokia N800				Nokia 770			
Storage location	INS	SEL	UPD	DEL	INS	SEL	UPD	DEL	INS	SEL	UPD	DEL
data stored in RAM	289	5	372	47	350	6	353	57	427	7	412	57
Flash internal	462	6	618	37	402	6	411	58	481	7	504	56
/dev/mmc2 (internal)	496	6	459	38	991	9	1112	55				
/dev/mmc1 (miniSD)	824	6	939	44	713	7	799	54				
/dev/mmc1 (1GB MMCM)					535	7	555	49	673	7	691	59

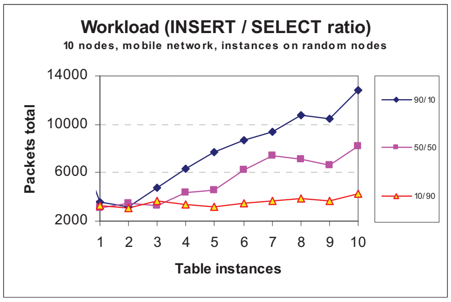
4.10.2 Choice of Java Interpreter

Only a few Java interpreters were available for the target devices, each with their own performance, features and issues. *JamVM* and *Cacao* were the ones supported by the Nokia Internet tablets, where Cacao proved to have a much shorter start-up time than JamVM. However, both were used interchangeably during the project's duration, due to various issues and limitations encountered in each of them. On the Zypad device, also tested briefly, the *J9* interpreter was the only one available at the time.

4.10.3 Packet Loss as a Result of Route Changes

In Section 12.4.4, we present results from experiments on an emulated mobile network, with focus put on applications' experience of packet loss, and ignoring packet loss at the synchronization components (GMDM and DS). Here, we show results experienced by the MDS component, which include synchronization messages as well.

We ran a dynamic scenario having 10 nodes, each sending 1 query per second. There were 47 neighbor changes during the 60 s of the scenario, causing a total of 342 route changes. The inertness of the underlying routing protocol to react to changes caused certain packet loss. Figure 4.6 shows that the percentage of lost packets is mostly influenced by the applications' access pattern to the database (i.e., the *write* to *read*) ratio, while the amount of replicas in the network does not



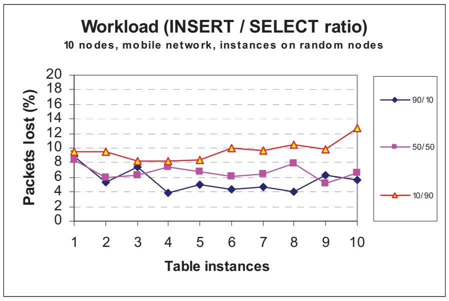


Figure 4.6: Total number of packets vs. lost packets

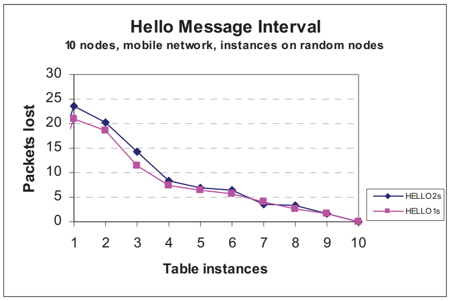


Figure 4.7: Impact of Hello Message interval on packet loss

play a big role. This is valid for each of the three ratios, although we can see that the percentage increases with the increase of *read* operations. This can be explained by the relatively low amount of original packets, and a more or less constant amount of synchronization packets, and thus lost packets.

In OLSR, *Hello* messages are used for neighbor detection. If a node fails to receive a certain number of consequent *Hello* messages (the proposed value is 3, i.e., 6 seconds) from one of its neighbors, the neighboring node is removed from the list of neighbors. As a consequence, there might be stale routes towards this node in the given period causing loss of packets. We have tried to reduce the Hello message interval from 2 to 1 second (and, consequently, the timeout period from 6 to 3 seconds), to see the impact this would have on the number of lost messages. Contrary to our expectations, Figure 4.7 shows that no significant gain has been achieved in our scenario. The gain is minimal, and in the cases where 50 % or more nodes host a replica of the table, the results for both cases are virtually the same.

4.11 Final Notes on MDS

In this section, we briefly present some of the remaining features and issues addressed by the MDS.

Clock Synchronization

Clock synchronization and issues tied to it have been a hot topic in the MIDAS project. It is clear that in disruptive networks, such as the ones MIDAS is targeting, it is unreasonable to expect all nodes to have perfectly synchronized clocks. Thus, some other mechanisms need to be deployed to avoid this fact causing problems such as synchronization loops.

In MDS, we do not rely on perfectly synchronized clocks, but we do assume that the clocks are synchronized before the event takes place, and that they will not skew noticeably by the end of the event. Under these assumptions, MDS can perform automatic conflict resolution by comparing record timestamps to determine which of the records is considered to be *newest*. Even though this might not be perfectly correct with respect to what is newest in reality, it will always give an unambiguous result at any of the nodes doing the comparison, which is needed to avoid endless loops that might be present if nodes were not able to agree.

Another approach, presented in [10], introduces the notion of *arrogant clocks* to achieve a common notion of time between nodes in the same network. In this system, each node thinks its own clock is perfect, and it monitors time differences between itself and other nodes. To achieve a common notion of time, timestamps on synchronized records are always updated to the local time on the node receiving the records.

Storing Binary Objects

To be able to store binary objects (so called *blobs*) into the database, applications must encode them into strings using an algorithm such as *Base64* (and decode them afterwards to read the values). Such functionality is provided directly by MDS. It has to be noted that, by doing this, the size of such objects increases by a constant factor of 1.3x.

Greylisting

To try to reduce unnecessary timeouts due to nodes being temporarily unavailable, MDS implements a *greylisting* scheme. Nodes that fail to respond to remote queries are greylisted locally. When MDS decides where a remote query should be sent, greylisted nodes will be put at the end of the list returned by GMDM, thus avoiding (if possible) timeouts from nodes that were detected to be out of reach. Optionally, it may be chosen on start that such nodes will be completely excluded from the list (i.e., they can be *blacklisted* instead). When an INSERT is issued, if there are no nodes within reach having a replica of a specific table, this will cause a local instance to be created. Whenever a message is received from a node that had been greylisted, they are removed from the greylist.

Packet compression and fragmentation

The proof-of-concept implementation of MDS includes automatic compression of packets. Since messages are sent as Java objects, which often include plain text, this allows for savings. Fragmentation of packets is also done automatically, to ensure that even large messages reach their destination.

These features should ideally be available by the underlying CRT component, but were for practical reasons related to project development implemented in MDS instead.

Chapter 5

Overview of the Research Papers

This thesis comprises seven research papers covering three different, yet strongly related categories within the main topic. Here we present a short synopsis of these research papers.

In emergency and rescue operations, where human lives are at stake, the main focus must be put on fast, efficient and reliable data sharing. Mobile ad-hoc networks (MANETs) can provide a platform for efficient data sharing, but in order to ensure availability and reliability, special mechanisms need to be present. In addition, the broadcast nature of wireless networks poses security risks that would not be present (at least not to the same extent) if cables were used.

The first research paper summarizes and describes the main security issues present in the target scenario. Some of the issues include securing data exchange (authentication, integrity, confidentiality), with special focus put on routing traffic which, if left unsecured, can lead to fatal network disruptions. The next research paper presents a concrete solution to the latter issue, a protocol used for securing the traffic at the network layer, thus providing the upper layers with a working network.

For the purpose of developing and testing this protocol, as well as other protocols described in later papers, we developed an emulation environment. This environment, described in the third research paper, was a more practical (and cheaper) alternative to performing pure simulations on one side, or real world experiments on the other side.

With a secured network in place, focus could be shifted to the main topic, namely data sharing in MANETs. In order to achieve a reliable service, which is resilient to network partitions or single nodes disappearing, there should be in the network more than one copy of each data entity. However, this comes with a price, that of transferring copies of data to different locations, and keeping track of all the replicas. The fourth paper describes how topology information extracted from the routing protocol can be exploited to optimize various synchronization techniques, while the fifth paper focuses on the metadata management used by the Midas Data Space (MDS) to keep track of the actual location of table replicas is stored on each node.

The sixth paper presents the first MDS prototype, including a basic evaluation. Finally, the last paper shows results of a detailed evaluation of different replica placement strategies that will be used as basis for our future work in optimizing MDS.

5.1 Paper #1

Title: Security and Privacy Issues in Middleware for Emergency and Rescue Applications

Authors: Matija Pužar, Thomas Plagemann, Yves Roudier

Publication: The 1st International Workshop on MObile and DIstributed approaches in Emergency Scenarios (MODIES 2008), Tampere, Finland, January 2008 [43]

Abstract: Mobile ad-hoc networks (MANETs) are a natural candidate for communication and information exchange in emergency and rescue operations. The personnel's movements, network disruptions and other system dynamics make it hard to implement robust applications for such environments. The MIDAS project aims at creating a middleware platform to simplify the task of developing and deploying mobile and robust services for events in which the network might be set-up at short notice. MANETs may be used because infrastructure is non-existing, and the number of users might be very high. One of the application domains addressed by MIDAS are emergency and rescue operations. To get a broad acceptance of the MIDAS solutions, security and privacy issues need also to be addressed. In this paper, we analyze the security threats and present a two-way approach to securing the MIDAS architecture. In the bottom-up approach, we use an efficient key management protocol to establish trust, and in the top-down approach we use dynamic role based access control to secure the system and provide privacy.

5.2 Paper #2

Title: SKiMPy: A Simple Key Management Protocol for MANETs in Emergency and Rescue Operations

Authors: Matija Pužar, Jon Andersson, Thomas Plagemann, Yves Roudier

Publication: The 2nd European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2005), Springer-Verlag, LNCS 3813, Visegrad, Hungary, July 2005 *(also published at UiO as Technical Report #319, ISBN 82-7368-272-2, February 2005)* [10]

Abstract. Mobile ad-hoc networks (MANETs) can provide the technical platform for efficient information sharing in emergency and rescue operations. It is important in such operations to prevent eavesdropping, because some the data present on the scene is highly confidential, and to prevent induction of false information. The latter is one of the main threats to a network and could easily lead to network disruption and wrong management decisions. This paper presents a simple and efficient key management protocol, called SKiMPy. SKiMPy allows devices carried by the rescue personnel to agree on a symmetric shared key, used primarily to establish a protected network infrastructure. The key can be used to ensure confidentiality of the data as well. The protocol is designed and optimized for the high dynamicity and density of nodes present in such a scenario. The use of preinstalled certificates mirrors the organized structure of entities involved, and provides an efficient basis for authentication. We have implemented SKiMPy as a plugin for the Optimized Link State Routing Protocol (OLSR). Our evaluation results show that SKiMPy scales linearly with the number of nodes in worst case scenarios.

5.3 Paper #3

Title: NEMAN: A Network Emulator for Mobile Ad-Hoc Networks

Authors: Matija Pužar, Thomas Plagemann

Publication: The 8th International Conference on Telecommunications (ConTEL 2005), Zagreb, Croatia, June 2005 *(also published at UiO as Technical Report #321, ISBN 82-7368-274-9, March 2005)* [42]

Abstract: Development of applications and protocols for wireless ad-hoc networks has always been a challenge. Specific characteristics such as frequent topology changes due to nodes moving around, popping up or being turned off, need to be considered from the earliest stages of development. Since testing and evaluation using genuine wireless devices is both expensive and highly impractical, other tools need to be used in the development phase. Simulators give a very detailed model of lower layers' behaviors, but code often needs to be completely rewritten in order to be used on actual physical devices. Emulators present a trade-off between real test beds and simulators, providing a virtual wireless network at the lowest layers, and yet allowing real code to be run on the higher layers. In this paper, we present such an emulation platform, called NEMAN, that allows us to run a virtual wireless network of hundreds of nodes on a single end-user machine. NEMAN has shown to be an important and very useful tool during development of different applications and protocols for our project, including a key management protocol and a distributed event notification service.

5.4 Paper #4

Title: Cross-layer Overlay Synchronization in Sparse MANETs

Authors: Thomas Plagemann, Katrine Stemland Skjelsvik, Matija Pužar, Aslak Johannessen, Ovidiu Valentin Drugan, Vera Goebel, Ellen Munthe-Kaas

Publication: The 5th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2008), Washington DC, USA, May 2008 [38]

Abstract: Mobile Ad-Hoc Networks maintain information in the routing table about reachable nodes. In emergency and rescue operations, human groups play an important role. This is visible at the network level as independent network partitions which are for some time stable before their members change through merging or partitioning. We use the information from stable routing tables to optimize the synchronization of Mediators in a Distributed Event Notification System. In a stable partition each node has the same information, thus a single Mediator can efficiently coordinate the synchronization, while all other Mediators just receive updates. We show in our experiments that just a few seconds are needed until routing tables stabilize and all nodes have a common view of the partition. We present a heuristic to determine the proper time to synchronize. Furthermore, we show how exceptions, like disappearing coordinating Mediators and unexpected messages, can be efficiently handled.

5.5 Paper #5

Title: Information Sharing in Mobile Ad-Hoc Networks: Metadata Management in the MIDAS Dataspace

Authors: Ellen Munthe-Kaas, Aslak Johannessen, Matija Pužar, Thomas Plagemann

Publication: The 10th International Conference on Mobile Data Management: Systems, Services and Middleware, Taipei, Taiwan, May 2009 [32]

Abstract: An approach to information sharing in mobile adhoc networks (MANETs) is to store on every node a small amount of metadata describing what information resources exist in the network and where they reside, allowing applications to first search locally for information about suitable resources, and next request relevant information from a node that contains the resource. The characteristics of MANETs in general and sparse, delay-tolerant networks in particular, make the task of maintaining and disseminating metadata across all nodes difficult, particularly in the presence of scarce resources. We have designed and implemented three protocols which use different approaches to metadata dissemination: epidemic one-to-one routing, one-to-many broadcasting that utilises the characteristics of the shared radio medium of wireless networks, and a protocol where a group consisting of e.g. rescue team leaders is given priority in the dissemination process and afterwards serves as multiple starting points for further dissemination.

In the MIDAS project we aim to produce middleware that speeds up MANET application development. By prototyping the metadata management component of MIDAS and testing the prototype with each of the three protocols in the network emulator environment NEMAN, we have measured bandwidth usage and performance. The implemented broadcast protocol is measured to use substantially less bandwidth than epidemic routing. The group protocol shows that the group indeed gets priority; it is comparable to the broadcast protocol in terms of bandwidth usage. The broadcast protocol has been used successfully in field tests of the MIDAS middleware.

5.6 Paper #6

Title: Information Sharing in Mobile Ad-Hoc Networks: Evaluation of the MIDAS Data Space Prototype

Authors: Matija Pužar, Katrine Stemland Skjelsvik, Thomas Plagemann, Ellen Munthe-Kaas

Publication: The 2nd International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS 2009), Toronto, Canada, June 2009 [44]

Abstract: Information sharing in dynamic mobile ad-hoc networks is a challenging task. High data availability in the presence of short and long term disconnections can be obtained by replicating shared data. The number of replicas must however be balanced against the cost of consistency management. In the MIDAS Data Space (MDS) we use optimistic replication together with internal versioning of data; this allows application-specific conflict resolution when reconciling replicas at network mergings. We have made a proof-of-concept implementation to perform experiments and to demonstrate through real-life field tests the usefulness of our design. In this paper we report our results. We have conducted a number of experiments on a small network formed by real devices to obtain a detailed performance evaluation. Using an emulation environment we have analysed and quantified the cost of consistency management, the impact of MDS operations, and the relationship between data availability and replication.

5.7 Paper #7

Title: Evaluation of Replica Placement Strategies for Mobile Ad-Hoc Networks

Authors: Matija Pužar, Thomas Plagemann

Publication: The 13th International Conference on Network-Based Information Systems (NBiS-2010), Takayama, Gifu, Japan, 2010 [41]

Abstract: The dynamic nature of mobile ad-hoc networks (MANETs) can easily lead to data being inaccessible due to constant route changes and network partitions. One method often used for increasing reliability and availability of data is replication. However, replication comes with costs, those of transferring and storing data and keeping track of consistency between replicas. For that reason, we have identified the core factors impacting the resulting network traffic. We have performed experimental studies with a real world prototype of a distributed data management system for MANETs. Furthermore, we have done an extensive simulation study showing where table replicas should be placed in the network, in order to minimize network traffic generated by access to the databases and by the synchronization data. The results of the experiments are consistent and show that by using clustering techniques we can achieve close-to-optimal traffic by placing replicas on approximately 10 % of nodes.

5.8 Other Publications

In addition to the abovementioned papers, this thesis has also contributed to the following publications.

- Thomas Plagemann, Ellen Munthe-Kaas, Katrine S. Skjelsvik, Matija Pužar, Vera Goebel, Ulrik Johansen, Joe Gorman, and Santiago Perez Marin, *A Data Sharing Facility for Mobile Ad-Hoc Emergency and Rescue Applications*, Proceedings of the First International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS 2007), Toronto, Canada, June 2007 [37]
- Norun Christine Sanderson, Katrine Stemland Skjelsvik, Ovidiu Valentin Drugan, Matija Pužar, Vera Goebel, Ellen Munthe-Kaas, Thomas Plagemann, *Developing Mobile Middleware - An Analysis of Rescue and Emergency Operations*, Research Report #358, ISBN 82-7368-316-8, ISSN 0806-3036, June 2007 [48]
- Erek Göktürk, Matija Pužar, Naci Akkøk, *Distributing NEMAN Network Emulator Using MICA Component Architecture*, Proceedings of the International Modeling and Simulation Multiconference 2007 (IMSM07) AIS-CMS, Buenos Aires, Argentina, February 2007 [13]
- Munthe-Kaas et al., *Mobile Middleware for Rescue and Emergency Scenarios*, The Handbook of Mobile Middleware (chapter), P. Bellavista and A. Corradi, Auerback Publications, 2006 [31]
- Plagemann et al, *Middleware services for information sharing in mobile ad-hoc networks - challenges and approach*, Workshop on Challenges of Mobility, IFIP TC6 World Computer Congress, Toulouse, France, August 2004 [39]

Chapter 6

Conclusions and Future Work

In this chapter, we present a short summary of the thesis, followed by a critical review of the claims presented in the introduction. Finally, we present some open problems and future work.

6.1 Summary

Due to the dynamic nature and unpredictable location of emergency and rescue operations, MANETs pose as a natural candidate for data sharing among people and organizations involved. The Ad-Hoc InfoWare [39] and MIDAS [9] projects look deeper into how MANETs can be used in such situation, and identify some important issues that would not be present if traditional static networks could have been used. In Chapter 3, we present a detailed requirements analysis of emergency and rescue operations, including examples of three different scenarios, and a description of how MANETs can be used in these.

The broadcast nature of wireless networks introduces specific challenges with respect to security, compared to wired networks. In wired networks, traffic can be sent through the wires without worrying about unauthorized access, as long as their physical access is strictly controlled. In wireless networks, however, anyone in the near vicinity can eavesdrop on traffic and, what is in some cases even more dangerous, anyone can generate traffic and induce it into the network. In emergency and rescue operations, where human lives depend on the operation's success, it is of outmost importance to make sure that only authorized devices can take part in the network, to prevent intruders from disrupting the network. In Chapter 7, we present a general overview over security issues in MANETs for emergency and rescue operations, while Chapter 8 focuses on the particular issue of network access, and describes a key exchange protocol that can be used on the network layer to protect the traffic, and to ensure that intruders are being kept out.

The dynamic nature of wireless networks, especially mobile ones, introduces challenges with respect to data sharing. In MANETs, it is not uncommon that nodes constantly join and leave the network, or that routes between existing members change. Frequent topology changes and disappearing nodes can have a negative impact on data availability. In emergency and rescue operations, where data availability is a critical factor, mechanisms must be present to cope with topology changes. In Chapters 4, 11 and 12, we present in detail the design and the first prototype of the MIDAS Data Space (MDS). MDS is a shared data space designed for such dynamic networks, ensuring data availability by means of replication, and eventual consistency by means of data versioning. In Chapter 13, we show how replication can be performed dynamically by monitoring both the applications' usage of MDS, and the network topology. We present also an extensive study of replica placement in the cases where these two factors are unknown upfront.

Developing applications and protocols requires extensive testing and evaluation procedures before they are ready to be deployed to target devices. When target devices are numerous and mobile, like the ones forming MANETs, the cost of using real devices during the development process is too high. One must have in mind the amount of people needed to participate, the time needed for test runs and deployment, the fact that experiments need to be re-run when bugs are fixed or changes are made, etc. For that reason, simulators or emulators are the preferred method to be used during the initial phases of development. While simulators can give a more realistic networking layer and better reproducibility, they often require applications and protocols to be developed especially for the particular simulator being used. This in turn can induce new bugs when these are rewritten in order to be deployed on real devices. Emulation, on the other hand, allows for real code to be run on top of a simulated network, considerably reducing the time needed to implement an application or protocol, with the possibility to later install them to target devices with minor changes or no changes at all. As part of this thesis, we created an emulation platform (see Chapter 9 for details) to facilitate our study and development, testing and evaluation of various applications and protocols for MANETs.

6.2 Critical Review of Claims

In Section 1.3, we presented the main contributions of this thesis in form of four claims. This section revises these claims, each followed by a critical review.

Claim 1:

In emergency and rescue operations, security has to be present at the lowest layers, in order to prevent malicious nodes from disrupting the network. The hierarchical organization of the entities involved in these kinds of operations can provide a basis for trust establishment between rescue personnel's devices. We provide a solution that exploits this fact in order to agree on a shared key. The solution can be used to ensure that intruders cannot disrupt the network by injecting false routing information.

Chapter 7 presents a detailed analysis of security requirements for MANETs in emergency and rescue operations. In Chapter 8, we present SKiMPy, a key exchange protocol designed and optimized especially for highly dynamic ad-hoc networks, such as MANETs used in emergency and rescue operations. The protocol uses to its advantage the fact that organizations participating in emergency and rescue operations are known upfront, making it possible to pre-install certificates for efficient off-line cross-authentication of nodes on the spot. By doing this, only trusted nodes are able to take part in the network, thus preventing non-authorized nodes from malicious actions such as injecting false routing information. Even though the protocol achieves its main purpose, in its current form it lacks important features such as exclusion of compromised nodes, revocation of keys and certificates, duplicate key ID numbers, or protection from denial of service attacks. Since the protocols works above the physical layer, the protocol cannot protect from attacks such as signal jamming, where other methods need to be used.

Claim 2:
With a secured infrastructure, the system can be used for information sharing between the rescue personnel. A shared data space, such as MIDAS Data Space, can be used to efficiently and robustly distribute information, even on small resource-limited devices. The strict requirement of accounting for such operations can be achieved by means of versioning.

In Chapters 4 and 11, we present MDS, a shared data space designed to cope with dynamic environments, such as MANETs in emergency and rescue operations. In Chapter 12, we analyze the performance of MDS on resource-limited devices, followed by a more extensive evaluation of replica placement strategies in Chapter 13. By means of versioning, which allows for permanent storage of data and their change history, the rescue operation can in the post-processing phase be analyzed in detail. To achieve that, it is assumed that devices have enough storage place. One clear limitation of such a system is that it does not support real-time data, but rather only delay tolerant data. Also, the applications need to be aware that the system supports only eventual consistency, as strict consistency cannot be guaranteed (as explained in Section 2.1.3).

Claim 3:
It is possible to achieve a close-to-optimal placement of table replicas within the network, which works well for most scenarios where the applications' access pattern to the data space or the network topology are unknown a priori. At a later point, the system can adjust to the concrete situation by analyzing the applications' access pattern and network topology.

In Chapter 10, we show how synchronization protocols can benefit from a cross-layer approach, such as using topology information from the networking layer to plan actions on the application layer. In Chapter 13, we investigate this possibility further and, by means of extensive measurements of clustering methods, we show how a close-to-optimal placement of table replicas can be achieved without having information on access pattern to the data space, and by only analyzing the current network topology. Gaining knowledge of network topology requires both a proactive protocol and a means to communicate with it, limiting the choice of the routing protocols that can be used. Like any centralized solution, there is a single point of failure that needs to be taken care of. In this case, it is possible for any node to take over the role of Traffic Controller within a few moments, and to start gathering access statistics for future placement decisions. The overhead introduced by reporting access statistics to the Traffic Controller is minimal, but it does cause constant traffic as long as a table is being accessed.

Claim 4:
In order to develop and evaluate protocols, an adequate test-bed has to be in place. An emulation test-bed is the most flexible choice with respect to development time and costs, as well as choice of programming languages. We have implemented such an emulation test-bed and used it to develop, test and evaluate a variety of protocols.

Chapters 2 and 9 give an overview of what options one has for testing and evaluating protocols and applications for MANETs, and argue why emulation is our preferred method. The emulation test-bed NEMAN, described in Chapter 9, has been used actively in our research for the last few years, leading to numerous scientific papers (e.g., [12], [13], [32], [38], [40], to name a few). It has also been used in teaching courses, as well as part of several master's theses and doctoral dissertations (e.g., [11], [52], [2], [20], [22], [50]). The test-bed runs on a single PC, making the server's resources a bottleneck when simulating larger networks. It is also a problem if several CPU and memory intensive processes (such as Java programs) are being run concurrently. Some physical layer issues (e.g., grey zones, packet collisions) are supported, while others (most importantly bandwidth and delays) still need to be implemented.

6.3 Open Problems and Future Work

The Ad-Hoc InfoWare [39] and MIDAS [9] projects have identified and addressed a number of issues present when using MANETs in emergency and rescue operations. The projects resulted in four doctoral dissertations at the University of Oslo (i.e., [6], [49], [52], and the present one), each addressing a specific issue.

This thesis addresses three of those issues, namely security, data sharing, and testing and evaluation of protocols in MANETs. Within these topics, several problems still remain unsolved.

The protocol presented in Chapter 8 enables authorized nodes to establish a shared key. The protocol is based on the fact that devices have pre-installed certificates, allowing for a cross-authentication on the spot. However, if a device is lost or stolen (i.e., compromised), there must be a mechanism to ensure that such a device's certificate is revoked and, consequently, that the device does not have access to the network any more. In its current state, the protocol lacks such a mechanism for revocation of certificates, a non-trivial issue given the assumption that an on-line certification authority cannot be counted on. Authentication of users towards the devices, which is sufficiently user-friendly to be applicable in such a scenario (i.e., which does not hinder the rescue personnel in the rescue operation), is also a challenge.

The MIDAS Data Space (MDS), presented in Chapter 4, includes a working implementation of all its sub-components. Some of them, however, either have trivial functionality, or they might not be scalable. Initial work on a dynamic Data Allocator (DA) sub-component is presented in Chapter 13. The paper presents methods for minimizing synchronization traffic in dynamic environments where replication is necessary or desirable. The paper includes some ideas that should be looked deeper into, e.g., what gain can be achieved by using the described combination of warm standby replicas and caching of query results and synchronization traffic. Designing and implementing an algorithm that includes all the methods and ideas from Chapter 13 is not a trivial task and is part of future work.

Finally, it would be very interesting to verify all the results and ideas presented in this thesis on a large scale real world test-bed.

Bibliography

[1] Andersen, A. et al..,"Reflective Middleware and Security: OOPP meets Obol", in Proceedings of the 2nd Workshop on Reflective and Adaptive Middleware, Rio. June 2003

[2] Bjerve, E., "Transparent gateways between OLSR networks", Master's thesis, Department of Informatics, University of Oslo, 2006

[3] Blair, G.S. et al.,"The design and implementation of Open ORB 2", in IEEE Distributed Systems Online, 2(6), 2001

[4] Clausen, T., Jacquet, P., "Optimized Link State Routing Protocol (OLSR)", RFC 3626, 2003

[5] Devlić, A., Koziuk, M., Horsman, W., "Synthesizing context for sports domain on a mobile device", The 3rd European Conference on Smart Sensing and Context (EuroSSC 2008), Springer-Verlag LNCS 5279, pp. 206-219, Zurich, Switzerland, 2008

[6] Drugan, O.V., "A Communication Non-Intrusive Middleware for Resource Management in Sparse Mobile Ad-Hoc Networks", PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, nr. 741, Norway, 2008

[7] Earthquake scenario for western Nevada, November 1996, http://www.nbmg.unr.edu/dox/nl/nl30.htm

[8] Gilbert, S., Lynch, N., "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services", ACM SIGACT News, vol.33, nr.2, 2002

[9] Gorman, J., "The MIDAS Project: interworking and data sharing", Interworking 2006, Santiago, Chile, January 2007

[10] Gorman, J., Wienhofen, L.W.M., "Common Notion of Time in the MIDAS MANET: Arrogant Clocks", Norsk Informatikkonferanse NIK 2008, Kristiansand, Norway, 2008

[11] Göktürk, E., "MICA: A Minimalistic Component-Based Approach to Realization of Network Simulators and Emulators" , PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, nr. 653, Norway, 2007

[12] Göktürk, E., "A Stance on Emulation and Testbeds, and a Survey of Network Emulators and Testbeds", Proceedings of the 21st European Conference on Modelling and Simulation (ECMS 2007),

[13] Göktürk, E., Pužar, M., Akkøk, N., "Distributing NEMAN Network Emulator Using MICA Component Architecture", Proceedings of the International Modeling and Simulation Multiconference 2007 (IMSM07) AIS-CMS, Buenos Aires, Argentina, February 2007

[14] Grace, P., Blair, G. S. and Samuel, S., "Interoperating with Services in a Mobile Environment", Technical Report (MPG-03-01), Lancaster University, 2003

[15] Hafslund, A., Tønnesen, A., Rotvik, J.B., Andersson, J., Kure, Ø., "Secure Extension to the OLSR protocol", OLSR Interop Workshop, San Diego, August 2004

[16] Hollick, M., Schmitt, J., Seipl, C., Steinmetz, R., "On the Effect of Node Misbehavior in Ad Hoc Networks", Proceedings of IEEE International Conference on Communications, ICC'04, Paris, France, volume 6, pages 3759-3763. IEEE, June 2004

[17] HyperSQL Database, http://www.hsqldb.org/

[18] H2 Database Engine, http://www.h2database.com/

[19] Isabelli, M., "Application Scenario – Emergency", MIDAS Project Deliverbale D5.2, 2006

[20] Jama, S.H., "Revising the User Interface of NEMAN", Master's thesis, Department of Informatics, University of Oslo, 2007

[21] Johnson et al., "Mobile Emulab: A Robotic Wireless and Sensor Network Testbed", Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006), Barcelona, Spain, 2006

[22] Johnsen, J.E., "Physical Layer Issues in NEMAN", Master's thesis, Department of Informatics, University of Oslo, 2006

[23] Johnson, D., Maltz, D., Hu, Y-C., "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks for IPv4", RFC 4728, 2007

[24] Kärpijoki, V., "Security in Ad Hoc Networks", Tik-110.501, Seminar on Network Security, HUT TML, 2000

[25] Kropff, M., Krop, T., Hollick, M., Mogre, P.S., Steinmetz, R., "A survey on real world and emulation testbeds for mobile ad hoc networks", in Proceedings of TRIDENTCOM'06. IEEE, March 2006

[26] Kiczales, G. et al., "Aspect-oriented programming", in Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), LNCS 1241, Springer Verlag, pp. 220-242, Jyväskylä, Finland, June 1997

[27] Koziuk, M., Domaszewicz, J., Schoeneich, R.O., "Mobile Context-Addressable Messaging with DL-Lite Domain Model", The 3rd European Conference on

Smart Sensing and Context (EuroSSC 2008), Springer-Verlag LNCS 5279, pp. 168-181, Zurich, Switzerland, 2008

[28] Lin, G., Noubir, G., "On Link Layer Denial of Service in Data Wireless LANs", Wireless Communications & Mobile Computing, vol. 5, nr. 3, pp. 273-284, May 2005

[29] Maes, P., "Concepts and experiments in computational reflection", in OOP-SLA'87: Conference proceedings on Object-oriented programming systems, languages and applications, ACM Press, Orlando, Florida, USA, 1987

[30] Mahadevan, P., Rodriguez, A., Becker, D., Vahdat, A., "MobiNet: A Scalable Emulation Infrastructure for Ad Hoc and Wireless Networks", ACM SIGMOBILE Mobile Computing and Communications Review, vol.10, nr.2, 2006

[31] Munthe-Kaas, E., Drugan, O., Goebel, V., Plagemann, T., Pužar, M., Sanderson, N., Skjelsvik, K. S., "Mobile Middleware for Rescue and Emergency Scenarios, Mobile Middleware", Paolo Bellavista and Antonio Corradi, ed., CRCPress, 2006

[32] Munthe-Kaas, E., Johannessen, A., Pužar, M., Plagemann, T., "Information Sharing in Mobile Ad-Hoc Networks: Metadata Management in the MIDAS Dataspace", The 10th International Conference on Mobile Data Management: Systems, Services and Middleware, Taipei, Taiwan, May 2009

[33] MySQL, http://www.mysql.com/

[34] Padmanabhan et al., "A survey of data replication techniques for mobile ad hoc network databases", The VLDB Journal - The International Journal on Very Large Data Bases, v.17 n.5, p.1143-1164, August 2008

[35] Perkins, C., Belding-Royer, E., "Ad hoc On-Demand Distance Vector (AODV) Routing", RFC 3561, July 2003

[36] Perkins, C., Bhagwat, P., "Highly Dynamic Destination-Sequenced Distance Vector (DSDV) for Mobile Computers", Proc. of the SIGCOMM 1994 Conference on Communications Architectures, Protocols and Applications, pp 234-244, 1994

[37] Plagemann et al., "A Data Sharing Facility for Mobile Ad-Hoc Emergency and Rescue Applications, Proceedings of the First International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS 2007), Toronto, Canada, June 2007

[38] Plagemann et al., "Cross-layer Overlay Synchronization in Sparse MANETs", 5th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2008), Washington DC, USA, May 2008

[39] Plagemann et al, "Middleware services for information sharing in mobile ad-hoc networks - challenges and approach", Workshop on Challenges of Mobility, IFIP TC6 World Computer Congress, 2004, Toulouse, France, Aug. 2004

[40] Pužar, M., Andersson, J., Plagemann, T., Roudier, Y., "SKiMPy: A Simple Key Management Protocol for MANETs in Emergency and Rescue Operations", Proceedings of the 2nd European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2005), Springer-Verlag, LNCS 3813, Visegrad, Hungary, July 2005

[41] Pužar, M., Plagemann, T., "Evaluation of Replica Placement Strategies for Mobile Ad-Hoc Networks", The 13th International Conference on Network-Based Information Systems (NBiS-2010), Takayama, Gifu, Japan, 2010

[42] Pužar, M., Plagemann, T., "NEMAN: A Network Emulator for Mobile Ad-Hoc Networks", Proceedings of the 8th International Conference on Telecommunications (ConTEL 2005), Zagreb, Croatia, June 2005

[43] Pužar, M., Plagemann, T., Roudier, Y., "Security and Privacy Issues in Middleware for Emergency and Rescue Applications", First International Workshop on MObile and DIstributed approaches in Emergency Scenarios (MODIES 2008), Tampere, Finland, January 2008

[44] Pužar, M., Skjelsvik, K.S., Plagemann, T., Munthe-Kaas, E., "Information Sharing in Mobile Ad-Hoc Networks: Evaluation of the MIDAS Data Space Prototype", The Second International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS 2009), Toronto, Canada, June 2009

[45] Rapport fra Åsta ulykken: NOU 2000: 30, Åsta-ulykken, 4. januar 2000, ISBN 82-583-0543-3, Oslo, Norway, 2000

[46] Ratsimor, O. et al., "Allia: alliance-based service discovery for ad-hoc environments", in Proc. of the 2nd ACM Mobicom Int. Workshop on Mobile Commerce (WMC'02), Atlanta, GA, September, 2002

[47] Raychaudhuri et al., "Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols", Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2005), New Orleans, USA, 2005

[48] Sanderson et al, "Developing Mobile Middleware - An Analysis of Rescue and Emergency Operations", Research Report #358, ISBN 82-7368-316-8, ISSN 0806-3036, June 2007

[49] Sanderson, N.C., "Network Wide Information Sharing in Rescue and Emergency Situations", PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, nr. 720, Norway, 2008

[50] Spigseth, Ø, "Introducing Name Resolution into OLSR", Master's thesis, Department of Informatics, University of Oslo, 2008

[51] SQLite, http://www.sqlite.org/

[52] Skjelsvik, K.S., "A Distributed Event Notification Service for Sparse Mobile Ad-hoc Networks", PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, nr. 714, Norway, 2008

[53] Svagård, I., “Overall architecture of MIDAS”, MIDAS project deliverable D2.1, 2008

[54] Sørhus, K.B., “The Optimized Broadcast Manager for MANETs (OBBM)”, Master’s thesis, Department of Informatics, University of Oslo, 2009

[55] The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns/

[56] The Norwegian SAR Service,
English: http://odin.dep.no/filarkiv/183865/Infohefte_engelsk.pdf
Norwegian: http://odin.dep.no/filarkiv/183864/Infohefte_norsk-lang.pdf

[57] The ns-3 network simulator, http://www.nsnam.org/

[58] Tønnesen, A., “Implementing and extending the Optimized Link State Routing protocol”, http://www.olsr.org/, August 2004

[59] Vahdat, A., Becker, D. “Epidemic Routing for Partially Connected Ad Hoc Networks”, Technical Report CS-2000-06, Department of Computer Science, Duke University, 2000

[60] Varga, A., “The OMNeT++ Discrete Event Simulation System”, Proceedings of the European Simulation Multiconference (ESM 2001), Prague, Czech Republic, 2001

[61] Zec, M., “Virtualized network stack in FreeBSD –CURRENT”, Tutorial talk given on EuroBSDCon 2007, Copenhagen, Denmark, 2007

[62] Zec, M., Mikuc, M., “Operating System Support for Integrated Network Emulation in IMUNES”, Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure / ASPLOS-XI, Boston, October 2004

[63] Zhao, W., Ammar, M., Zegura, E. “A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Netowrks”, MobiHoc’04, Roppongi, Japan, 2004

[64] Zeng, X., Bagrodia, R., Gerla, M., “GloMoSim: a Library for the Parallel Network Simulation Environment”, Proceedings of the 12th Workshop on Parallel and Distributed Systems, 1998

Part II

Research Papers

Chapter 7

Security and Privacy Issues in Middleware for Emergency and Rescue Applications

Authors: Matija Pužar[1], Thomas Plagemann[1], Yves Roudier[2]

Affiliations: (1) Department of Informatics, University of Oslo
`{matija, plageman}@ifi.uio.no`

(2) Institut Eurécom, Sophia-Antipolis, France
`yves.roudier@eurecom.fr`

Publication: The 1st International Workshop on MObile and DIstributed approaches in Emergency Scenarios (MODIES 2008), Tampere, Finland, January 2008

Abstract: Mobile ad-hoc networks (MANETs) are a natural candidate for communication and information exchange in emergency and rescue operations. The personnel's movements, network disruptions and other system dynamics make it hard to implement robust applications for such environments. The MIDAS project aims at creating a middleware platform to simplify the task of developing and deploying mobile and robust services for events in which the network might be set-up at short notice. MANETs may be used because infrastructure is non-existing, and the number of users might be very high. One of the application domains addressed by MIDAS are emergency and rescue operations. To get a broad acceptance of the MIDAS solutions, security and privacy issues need also to be addressed. In this paper, we analyze the security threats and present a two-way approach to securing the MIDAS architecture. In the bottom-up approach, we use an efficient key management protocol to establish trust, and in the top-down approach we use dynamic role based access control to secure the system and provide privacy.

7.1 Introduction

The MIDAS project [4] aims at developing a middleware platform to simplify and speed up the task of developing and deploying mobile services for events in which the network is set-up at short notice. MANETs may be used because infrastructure is non-existing, and the number of users might be very large. Besides large sports events like Tour de France, one of the main scenarios for applications developed using the MIDAS platform are emergency and rescue operations, where MANETs are used by the rescue personnel. Such operations are very dynamic and unpredictable, both when it comes to personnel's movements and membership, causing network disruptions and frequent changes of nodes within it. Therefore, the project has put special emphasis on the development of a data space for sharing information and a context space for managing context data as reliably as possible, to ensure the high availability of services and data, and their graceful degradation in case of long term network partitioning. While these non-functional requirements are essential for emergency and rescue applications, security and privacy also need to be addressed in order to get a broad acceptance of the project results for the application domain of emergency and rescue.

This paper presents a first conceptual study of how security and privacy can be supported within MIDAS and which security and privacy issues cannot be solved through middleware. These solutions are especially tailored for the emergency and rescue domain, but not necessarily for MIDAS only. Most of the concepts could also be applied to other distributed software of the same domain running over MANETs.

The core idea of our design is to combine bottom-up and top-down approaches. The bottom-up approach is based on earlier research results, namely the design of an efficient key management protocol, which enables us to efficiently agree on a shared key to achieve trust among the devices of emergency and rescue personnel. The top-down approach relies on the fact that roles typically define the duties and rights of rescue personnel. Due to the dynamics of rescue operations, individuals might change roles during the operation, which highlights the need for a dynamic role based access control system.

The remainder of this paper is structured as follows: Section 7.2 briefly describes the MIDAS architecture and Section 7.3 the basic assumptions for our solution. A summary of the key management protocol is given in Section 7.4. Section 7.5 introduces dynamic role based access control. Section 7.6 describes how these two ingredients work together and provide security and privacy in the MIDAS middleware. Conclusions are given in Section 7.7.

7.2 The MIDAS Architecture

The MIDAS Architecture is shown n Figure 7.1. The main components in MIDAS are MIDAS Data Space (MDS), Context Space (CXS) and Communication and Routing (CRT). Access Control (AC) and Network Security (NS) are new security components that are the contribution of this paper.

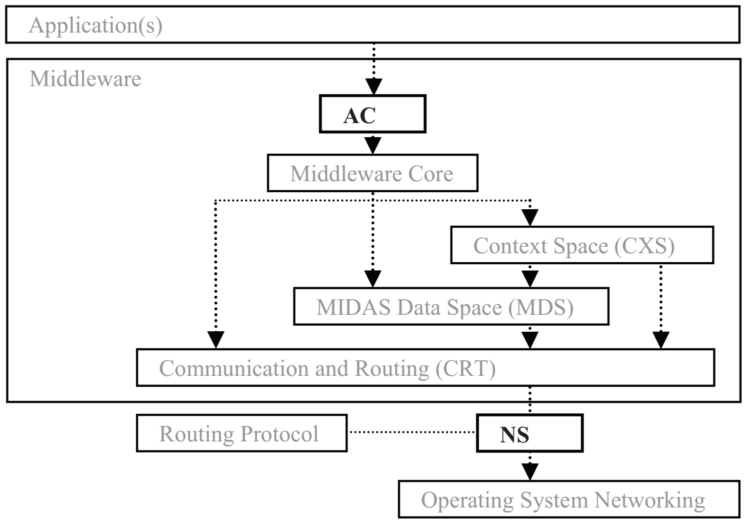


Figure 7.1: MIDAS Architecture

MDS is a relational data space provided to applications and other MIDAS components, shared between all the nodes [9]. The users and applications do not know where exactly in the network data are stored and they only see one single large data space. The MDS component does this by hiding the physical location of data from the applications, as well as by taking care of replicating data when there might be possibility for network partitioning. Replication of data to other nodes having a replica of the same table is done either eagerly (i.e. as soon as a change is done) or lazily (for nodes joining the network partition at a later point). From the application developers' point of view, only SQL queries have to be issued in order to use the shared data space; data are published to the data space from where they can later on be retrieved.

The CXS component is responsible for defining and managing all context data in the context space. It provides middleware support for domains by implementing domain-related middleware services. As each MIDAS mobile service targets events of a specific domain (emergency operations, sports events), the support of domains helps the developers to create very focused applications within their domain. CXS allows for simple or complex (synthesized) queries and uses MDS to store and retrieve all context information. In addition, the component offers the applications services for the context addressable messaging, where message recipients are defined as a context (for instance, "all nodes with role MEDIC that still do not have a patient assigned").

The CRT component is responsible for providing connectivity between nodes running the MIDAS middleware. Networks supported by CRT may be disruptive and even consist of different network technologies, which should all be seamless from the applications' and other components' point of view. CRT has mechanisms to allow connectivity even in cases when nodes are not in direct range of each other. If two network partitions have each a connection to the Internet (for instance through a GPRS connection), they can register at a pre-defined central node. As a consequence, they can merge into a single network by establishing a link between each other through the Internet.

Applications connect to the middleware using the Java Remote Method Invocation (RMI), in which the core middleware component acts as a façade towards its components, i.e., it internally performs calls to the requested methods in the respective components. Internally, components use each other's services either by direct calls or through the core component.

7.3 Basic Assumptions and Our Approach

There are five basic assumptions that were taken into account when designing security solutions for the MIDAS Architecture.

- Members of the rescue personnel (users) can be trusted.
- Devices are preconfigured before coming to the scene; this includes installation of the necessary software, certificates, etc.
- Users carry tokens with personal certificates containing their credentials (name, rank / role within the organization, etc.) and use them to authenticate themselves towards the device.
- The user-to-device authentication process is assumed to be secure enough
- Software developed on top of the MIDAS platform is considered clean (that is, it is not vulnerable, there are no viruses, trojans, etc.)

The overall security of the system depends on these assumptions to be satisfied beforehand.

The wireless medium is especially vulnerable to attacks and intrusions by malicious persons with the goal to either just disrupt the functionality of the network or to steal sensitive data, from health information about patients to possible police records. External attackers, however, are not the only issue. Not all the personnel are supposed to have access to all the data within the network. Policemen should, for example, not be able to access patients' medical records (privacy). Following the development of the situation, the personnel's roles and assignments might also change dynamically, providing additional challenges to the component enforcing access control (see Section 7.6 for details). Since accidents naturally happen where they are least expected or they might in severe cases cause existing communication infrastructure to collapse, nodes may be left with no reliable possibility of contacting a central authority to, for instance, check a user's credentials (authentication). Next, it must be ensured that unauthorized nodes cannot modify existing traffic or introduce new data into the network (integrity). Finally, it must be ensured that receipts of given orders cannot be denied at a later point (non-repudiation).

We present a two-way design towards solving the abovementioned security issues, consisting of a bottom-up and a top-down approach.

The bottom-up approach ensures that data coming from the network are authenticated and that their content's integrity is verified already at the network layer. The whole process is hidden from the users and no intervention is required from them. This is achieved by using a key management protocol such as SKiMPy [10] to ensure that only authorized nodes are allowed to participate in forming network. Failure to do so might result in a very small percent of misbehaving nodes being able to disrupt the whole network [5].

The top-down approach is used to determine what data, services and other resources a certain user has access to. The access control component does this by taking into account the user's identity and role when accessing the middleware.

There are still a number of issues to be taken care of, in order for the security level to be satisfactory. Some of these issues (such as denial of service attacks on the physical layer, devices being lost or stolen, etc.) cannot be solved by the middleware and must be addressed separately. Other issues can be solved by the middleware and, as such, they will be looked into as part of current and future work, including issuing temporary certificates, revocation of certificates for nodes being lost or stolen.

7.4 The Key Management Protocol

SKiMPy is a self-organizing distributed key management protocol used at the network layer to secure either all the data coming from and to the node, or only the routing protocol (such as OLSR [1]). SKiMPy uses certificates that are installed on the rescue personnel's devices in advance of the emergency event, i.e. before coming to the rescue scene. These authorized nodes can be called active members of the network. All the certificates are, at the top of the hierarchy, signed by the same Certificate Authority and can thus be cross-verified by any node without the need for contacting a central authority. That way, unauthorized nodes can be excluded from participation in all or some network activities, while trust can be achieved among arbitrary devices of rescue and emergency personnel without access to the Internet. Not having contact with a central authority leads to problems in case it is necessary to revoke certificates of devices that have been lost or stolen. If anyone could issue a certificate revocation, this could also be done by the same device that is supposed to be cut out of the network, causing a network breakdown. On the other hand, a voting procedure might take place, where for example k out of n signatures would be necessary to issue a valid revocation. While one could argue that the authentication of the user to the device provides a certain level of security to assure that lost or stolen devices cannot be misused, this problem needs still to be solved to provide full security.

Contrary to the main exclusion policy, it is sometimes useful to grant spectators or media the ability to help at an accident scene. Such nodes might then be issued temporary certificates on the spot to be able to exchange non-critical data (i.e., become a passive member of the network, yet unable to influence the core network's routing tables). Such temporary certificates are then used by the same key management protocol to establish a second shared key between the passive members. The certificates would be issued by authorized users, depending on the security policies. In addition to a much shorter validity time of credential, the abovementioned authorized users would become local authorities in charge of temporary certificates' revocation with much less likelihood of losing contact.

It has to be noted that the shared key achieved by using SKiMPy can only be used to verify a message's integrity, not its source. The same applies in the case when the key is used to encrypt data. That is, it can only be shown that a message has been sent by some authorized node, not which one exactly.

7.5 Dynamic Role Based Access Control

The distributed and shared nature of the MIDAS Data Space, explained in Section 7.2, can cause specific issues with regards to access control.

There exist several methods of implementing access control. In Discretionary Access Control (DAC [2]), users can grant or remove other users access to resources they themselves have access to (a good example for a DAC system is the UNIX file system). Due to the shared nature of MDS and the number of organizations involved in emergency and rescue operations, this method is not expressive or flexible enough. In Mandatory Access Control (MAC [2]), in contrast, resources have security labels, while users have security clearances, which then must be high enough to be able to access a given resource. Organizational hierarchy can be the starting point for defining Role Based Access Control (RBAC [3]) rules used to determine access privileges certain roles have towards certain objects. Having indeed hierarchical organizations involved in emergency and rescue scenarios, a RBAC solution, such as OrBAC [6], is a clear candidate in our case.

The users identify themselves towards the OS or towards the middleware directly, while their role or roles are, as a rule, predefined by a person's rank in the organizational hierarchy. These a priori roles are defined in the user's certificates. However, the users' roles can be dynamically changed during the rescue operation, depending for instance on the order in which personnel come to the scene. It should be possible to reflect these changes in the roles used for access control to data and resources. Since roles are predefined and embedded within certificates, they cannot be changed. One step towards a solution might be to use the ideas from [8], where delegation and supervision are introduced. However, additional mechanisms might be necessary, e.g. roles issued in separate temporary certificates (similar to the ones described in Section 7.4) that can be either used in addition to the preinstalled ones, or they might supersede them. Removing certain roles, on the other hand, is in general a non-trivial issue, and it must be ensured that a user cannot choose to ignore a newly installed certificate giving them fewer rights. However, we assume that the rescue personnel can be trusted and will therefore also not take countermeasures against those certificates.

In addition to the standard access control rules (e.g. "can read"), emergency and rescue operations might require a special type of rule, that is, "must read" and/or "confirm receipt"). A typical example would be an order issued by the rescue scene leader, whose receipt must be confirmed. Being signed with the recipient's certificate, this receipt cannot be repudiated at a later stage. Deontic logic introduces actions such as "permitted", "obliged", and "forbidden". A RBAC mechanism based on deontic logic, such as the one described in [7], might be a possible solution for fulfilling both of the abovementioned needs. However, it might also be possible to implement it in a simpler way, for instance by moving some of the extra functionality towards the application.

7.6 A Combined Solution

In this section, we outline how certificates and access control are used to secure the MIDAS middleware (see Figure 7.1). Being just a conceptual analysis, and due to space limitations, some details on particular solutions are not specified as of yet.

At the level of network interface to the operating system, all incoming and outgoing traffic must pass the Network Security Component (NS). NS should run separately from the middleware as it has to protect not only data going to and coming from not only the middleware, but also all signalling messages of the routing protocol. Outgoing traffic is signed with the shared key obtained through key management protocol. Incoming traffic is first checked for integrity with shared keys available at the recipient. With regards to the source's and recipient's type of membership, as described in Section 7.4, we have the following four cases:

1. The recipient is an active member
   a) The message is signed by an active member: all the traffic is let through
   b) The message is signed by a passive member: only non-routing messages selected according to the security policy are sent through
2. The recipient is a passive member
   a) The message is sent by an active member: all the traffic gets through
   b) The message is sent by a passive member: all the traffic gets through

In addition, the incoming traffic must be checked for replay-attacks, i.e. it must be ensured that a valid message cannot, at a later point, be re-sent by a malicious node. For instance, routing messages that are re-sent at a later point may cause the whole network to collapse.

The middleware's application programming interface (API) is designed so that all calls issued from the applications towards the middleware are passed through the Access Control (AC) component. It must be noted that all the inter-component calls must go through AC as well, since it could happen that a component tries to perform an illegal action. It is the task of AC to verify whether the given user has the necessary credentials for that call. In case it is a query towards the data space, the user must have the necessary rights to perform that query on the specified table or tables in the data space.

Remote queries have to be verified twice, first at the source node and then again at the destination node. This means that the issuer's credentials (including their public key) must be included in the message. The issuer's public key may be used to encrypt the response to the query, but a more light-weight solution might be preferred if such queries are expected to be performed often. The underlying shared data space could be used to distribute the credentials of all the users present at the scene, leaving up to MDS the task of keeping the information in the data space as consistent as possible.

Access rights are defined by the personnel's identities or roles, both of which are specified in their pre-installed certificates. To add or change roles dynamically, as a consequence of the rescue operation's development, personnel can be issued temporary certificates on scene. Given the possible severity of the situation, the notion of access rights should include access duties as well, meaning that some messages must be read by the personnel they are addressed to. That way, the receipt of an order cannot be denied at a later point, making it known who had been in charge of what at any time in the operation.

7.7 Conclusion

Reliability and availability are important non-functional properties for emergency and rescue applications. The MIDAS middleware provides these properties through optimistic replication. In order to achieve also security and privacy for MIDAS in emergency and rescue applications, we propose in this paper a combined bottom-up and top-down approach. The dynamic role based access control we propose comes very close to the way security levels and rights assigned to emergency and rescue personnel are assigned in the different organizations. Access control is enforced with the help of certificates. In addition to the misuse of middleware service, it is necessary to protect the network itself. Therefore, all traffic is screened at each node and all packets from non-trusted nodes are simply discarded to disable attacks on the network.

While this approach provides a high level of security for the network and the middleware and enables to protect data, there are still certain attacks that cannot be handled, especially attacks at the link layer, like noise etc.

Our ongoing and future work is concerned with two aspects. First, we aim to refine the proposed solution and prototype it to be able to perform evaluation testing of the presented solution. Next, we extend the solution also for other application domains, like large sports events, where existing infrastructure is an important part of the network.

Acknowledgments

This work has been performed in the context of the MIDAS project (funded by the European Commission's 6th Framework Program, Contract no. 027055), and the Ad-Hoc InfoWare project (funded by Norwegian Research Council IKT2010, Project No. 152929/431).

References

[1] Clausen, T., Jacquet P., “Optimized Link State Routing Protocol (OLSR)”, RFC 3626, October 2003

[2] Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC) (DoD 5200.28-STD 1985), Fort Meade, MD, Department of Defense, 1985

[3] Ferraiolo, D.F., Kuhn, D.R., “Role-Based Access Control”, Proceedings of the 15th National Computer Society Conference, National Institute of Standards and Technology, Gaithersburg, MD, October 1992, 554-563.

[4] Gorman, J., “The MIDAS Project: interworking and data sharing”, Interworking 2006, Santiago, Chile, January 2007

[5] Hollick, M., Schmitt, J., Seipl, C., Steinmetz, R., “On the Effect of Node Misbehavior inAd Hoc Networks”, Proceedings of IEEE International Conference on Communications, ICC’04, Paris, France, volume 6, pages 3759-3763. IEEE, June 2004

[6] Kalam, A. et al, “Organization Based Access Control”, 4th International Workshop on Policies for Distributed Systems and Networks (Policy 2003), Como, Italy, June 4-6, 2003.

[7] Kolaczek, G., “Application of deontic logic in role-based access control”, International Journal of Appllied Mathematics and Computer Science, vol. 12, no. 2, 2002, p. 269-275

[8] Moffett, J.D., Lupu, E.C., “The Uses of Role Hierarchies in Access Control”, Proceedings of the fourth ACM workshop on Role-based access control (RBAC ‘99), Fairfax, Virginia, 1999, p. 153-160

[9] Plagemann, T. et al, “A Data Sharing Facility for Mobile Ad-Hoc Emergency and Rescue Applications”, The First International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS 2007), Toronto, Canada, June 2007

[10] Pužar, M., Andersson, J., Plagemann, T., Roudier, Y., “SKiMPy: A Simple Key Management Protocol for MANETs in Emergency and Rescue Operations”, Proceedings of the 2nd European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2005), Springer-Verlag, LNCS 3813, Visegrad, Hungary, July 2005

Chapter 8

SKiMPy: A Simple Key Management Protocol for MANETs in Emergency and Rescue Operations

Authors: Matija Pužar[1], Jon Andersson[2], Thomas Plagemann[1], Yves Roudier[3]

Affiliations: (1) Department of Informatics, University of Oslo
`{matija, plageman}@ifi.uio.no`

(2) Thales Communications, Norway
`jon.andersson@no.thalesgroup.com`

(3) Institut Eurécom, Sophia-Antipolis, France
`yves.roudier@eurecom.fr`

Publication: The 2nd European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2005), Springer-Verlag, LNCS 3813, July 2005

Abstract. Mobile ad-hoc networks (MANETs) can provide the technical platform for efficient information sharing in emergency and rescue operations. It is important in such operations to prevent eavesdropping, because some the data present on the scene is highly confidential, and to prevent induction of false information. The latter is one of the main threats to a network and could easily lead to network disruption and wrong management decisions. This paper presents a simple and efficient key management protocol, called SKiMPy. SKiMPy allows devices carried by the rescue personnel to agree on a symmetric shared key, used primarily to establish a protected network infrastructure. The key can be used to ensure confidentiality of the data as well. The protocol is designed and optimized for the high dynamicity and density of nodes present in such a scenario. The use of preinstalled certificates mirrors the organized structure of entities involved, and provides an efficient basis for authentication. We have implemented SKiMPy as a plugin for the Optimized Link State Routing Protocol (OLSR). Our evaluation results show that SKiMPy scales linearly with the number of nodes in worst case scenarios.

8.1 Introduction

Efficient collaboration between rescue personnel from different organizations is a mission critical element for a successful operation in emergency and rescue situations. There are two central requirements for efficient collaboration, the incentive to collaborate, which is naturally given for rescue personnel, and the ability to efficiently communicate and share information. Mobile ad-hoc networks (MANETs) can provide the technical platform for efficient information sharing in such scenarios, if the rescue personnel is carrying and using mobile computing devices with wireless network interfaces.

Wireless communication needs to be protected to prevent eavesdropping. The data involved should not be available to any third parties, for neither publication or malicious actions. Another important requirement is to prevent inducing of false data. At the application layer this might for example lead to wrong management decisions. At the network layer it has been shown that a very few percent of misbehaving nodes easily can lead to network disruption and partitioning [17]. In both cases, efficiency of the rescue operation will be drastically reduced and might ultimately cause loss of human lives. In order to prevent such a disaster, all data traffic should be protected, allowing only authorized nodes access to the data. Given that devices carried by the rescue personnel will mostly have limited resources, any security scheme based solemnly on asymmetric cryptography will be too costly in terms of computing power, speed and battery consumption. Therefore, the use of symmetric encryption with shared keys is preferable for MANETs in emergency and rescue scenarios. Agreeing on a shared key in a highly dynamic and infrastructure-less MANET is a non-trivial problem and requires establishing trust relations between all devices. It is important for emergency and rescue scenarios that corresponding solutions are simple, efficient, robust, and autonomous. User interactions should be kept at an absolute minimum.

This paper describes a simple key management protocol, called SKiMPy, that can be used to establish a symmetric shared key between the rescue personnel's devices. By this, SKiMPy will set up a secure network infrastructure between authorized nodes, while keeping out unauthorized ones. It may be decided at the application layer whether the established shared key is robust enough for achieving some degree of data confidentiality as well. The basis for this simple and efficient solution is the fact that rescue personnel are members of public organizations with strict, well defined hierarchies. This hierarchy can be mirrored into a certificate structure installed a priori on their devices, i.e., before the accident or disaster actually happens. As a result, it is possible for the nodes during the rescue activity to authenticate each other on a peer-to-peer basis, without need for contacting a centralized server or establishing trust in a distributed approach.

The organization of the paper is as follows. Section 8.2 gives a detailed description of our protocol. In Section 8.3 we show some design considerations and respective solutions. Section 8.4 describes an implementation of the protocol together with evaluation results. In Section 8.5 we present related work. Finally, conclusion and future work are given in Section 8.6.

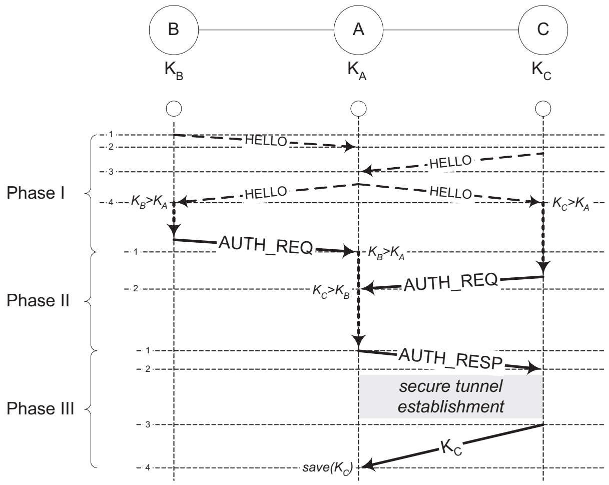


Figure 8.1: Message Flow Diagram

8.2 Protocol Description

SKiMPy makes use of the existing traffic in the network to trigger key exchange. Periodic routing beacons (HELLO), sent by proactive routing protocols, are such an example. The following two messages are specific to SKiMPy:

- *Authentication Request* (AUTH_REQ): sent by a node after it detects traffic from a node having a key that is *worse* than its own one. The message is used to inform the remote node that the sending node is willing to transfer its key.
- *Authentication Response* (AUTH_RESP): sent by a node, as a result of a re-ceived AUTH_REQ message. The message is used to inform the remote party that the node is willing to perform the authentication and receive the remote and *better* key.

The protocol consists of three phases, namely (I) Neighborhood Discovery, *(II) Batching* and *(III) Key Exchange*.

During phase I, a node listens to all traffic sent by its immediate neighbors. If it detects a node using a *worse* key (explained in detail in Section 8.3.2), it will send an *Authentication Request* message to it, saying it is willing to pass on its key. Upon receiving such a message, the other node enters the phase II, waiting for possible other authentication requests before sending a response. This batching period is used for optimization - a node will only perform authentication with the *best* of all neighbors. All the other keys will, due to the transitiveness property of the *better than* relation, at some point get overruled and therefore there is no point in getting them. After the node has chosen its peer, it sends an *Authentica-*

tion Response after which its peer initializes the actual authentication procedure, that is, exchange of certificates, establishing a secure tunnel, and finally transfer of the key. The reason for having such a hand-shake procedure is to ensure that the nodes can indeed communicate. In some standards, such as 802.11b [19], traffic like broadcast messages can be sent on a lower transmitting rate with larger transmission range than data messages. Thus, broadcast messages might reach a remote node and trigger a key exchange, even though the nodes cannot directly exchange data packets.

Figure 8.1 shows an example of the key exchange between three nodes (*A*, *B* and *C*) and indicates the different phases of the key exchange for node *A*. Node *A* enters phase I when turned on. Nodes *B* and *C* do not directly hear each other's traffic and are only able to communicate through node *A*, once the shared key is fully deployed.

The initial states of the three nodes are as follows: *A* has the key K_A, *B* has K_B and *C* has K_C. In this example, K_C is the *best* key, whereas K_A is the *worst* key.

Phase I:

1. Node *A* is turned on. All nodes send periodic HELLO messages which are part of the routing protocol.
2. *A* receives a HELLO message from *B*, notices a key mismatch, but ignores it because K_A is *worse* than K_B.
3. *A* receives HELLO from *C*, notices a key mismatch, but ignores it because K_A is *worse* than K_C.
4. *B* and *C* receive HELLO from *A*, they both notice they have a *better* key than K_A, and after a random time delay (to prevent traffic collisions), send an AUTH_REQ message to *A*.

Phase II:

1. A receives AUTH_REQ from *B* notices that *B* has a *better* key and schedules authentication with B. The authentication is to be performed after a certain waiting period, in order to hear if some of the neighbors has an even *better* key.
2. A receives AUTH_REQ from *C* as well, sees that *C* has a key *better* than K_B, and therefore decides to perform authentication with *C* instead.

Phase III:

1. *A* sends an AUTH_RESP message to *C*, telling it is ready for the authentication process
2. *C* initiates the authentication procedure with *A*, they exchange and verify certificates; the secure tunnel is established.
3. *C* sends its key K_C to *A* through the secure tunnel.
4. *A* receives the key and saves it locally; the old key K_A is saved in the key repository for eventual later use; *A* sends the new key further, encrypted with K_A.

In the next round, that is, after it hears traffic from node B signed with K_B, node A will use the same procedure to deliver the new key K_C to node B, hence establishing a common shared key in the whole cell.

There are two important parameters which influence the performance of the protocol and therefore have to be chosen carefully. The delays used before sending AUTH_REQ are random, to minimize the possibility of collisions in the case when more nodes react to the same message. On the other hand, the delay from the moment a node receives AUTH_REQ to the moment it chooses to answer with AUTH_RESP is a fixed interval and should be tuned so that it manages to hear as many neighbors as possible within a reasonable time limit. By this, all nodes that have been heard during the waiting period can be efficiently handled in the same batch.

8.3 Design Considerations

Our protocol is designed for highly dynamic networks, where nodes may appear, disappear and move in an arbitrary manner. Topology changes are inevitable. The key management protocol must have low impact on the available resources, i.e. battery, bandwidth and CPU time. Here, we analyze the different security and performance issues that had to be considered while designing the protocol, as well as respective solutions integrated into SKiMPy.

8.3.1 Authentication

An important characteristic of an emergency and rescue operation is that the organizations involved (police, fire department, paramedics, etc.) are often well structured, public entities. Before the rescue personnel comes to the disaster scene, all devices are prepared for their tasks. One task in the preparation phase, which we call *a priori* phase [33], is the installation of valid certificates. The certificates are signed by a commonly trusted authority, such as the ministry of internal affairs, ministry of defense, etc., on the top of the trust chain. This gives nodes the possibility to authenticate each other without need for contacting a third party.

Certificates on the nodes can identify devices, users handling them, or even both. The users would then present their certificate to the device by means of a token, i.e. smartcard. The decision for this does not impact the key management in SKiMPy, but it impacts the way how lost and stolen nodes are handled, i.e., revoking certificates and/or blacklisting of such nodes. We explain this issue later, in Section 8.3.5.

8.3.2 Choosing Keys

The main task of SKiMPy is to make sure that all the nodes agree on a shared key. When a node is turned on, it generates a random key with a random ID number. The uniqueness of the key IDs must be ensured by e.g. using the hash value of the key itself as part of the ID, by including the nodes MAC address, etc. The final shared key is always chosen from nodes' initial keys. To achieve this, we introduce the notions of *better* and *worse* keys, together with the relation "$>$" representing *better than*. There are several possible schemes for deciding which of the keys is *better* or *worse* and all schemes can be equally valid, as long as they cannot cause key exchange loops, are unambiguous and transitive: $(A > B$ and $B > C) => A > C$. The necessary control information, which depends on the scheme chosen, is always sent with the message signature.

We briefly describe two schemes and their advantages and drawbacks.

The first scheme uses arithmetic comparison of two numbers, i.e. the key having a higher or lower ID number, timestamp or a similar parameter, is considered to be *better*. The advantage of this scheme is that it is unambiguous, transitive and easy to implement. In addition, it can be "tweaked" in a way that would prevent a single node to cause re-keying of an already established network cell. For example, if the scheme defines that the lower ID number means a better key, the highest bit of the ID number can be always set to "1" when the node is turned on, and cleared once two nodes merge. Assuming that nodes in a certain area will in most cases pop up independently, this simple and yet efficient method might prevent a lot of unnecessary re-keying traffic. If we use the keys' timestamps instead of the ID numbers, choosing a lower timestamp could imply that the key is older and that more nodes have it already. SKiMPy does not require the clocks of different devices to be synchronized and therefore, the given assumption might not necessarily be true, especially if the key creator's clock was heavily out of sync. One major drawback of the presented scheme is that a small cell (consisting of, for example, 2 nodes) could easily cause re-keying of a much bigger cell (having, for example, 100 nodes), which would be a waste of resources.

The second scheme takes care of this problem by using the number of nodes in each network cell as the decisive factor. The simple rule for this scheme is to always re-key the smaller cell, i.e. the one with the lower number of nodes, thus minimizing resource consumption for the necessary re-keying. The approximate number of nodes can be either retrieved from the routing protocol state information (if, for example, the OLSR routing protocol [17] is used) or maintained at a higher protocol layer, as it is done in our project. However, if not all of the nodes have exactly the same information (which is to be expected in a dynamic scenario), and for some obscure reason we have more simultaneous merging processes between the same two cells, a key exchange loop may occur. One approach to this problem is to adjust in each node the state information of the number of nodes in its cell, always increasing it when new nodes join, but never decreasing it upon partitioning of the cell.

At the present, we use the first scheme, choosing always a key with a lower ID number. An in-depth study of both schemes and their variations is subject to ongoing and future work.

8.3.3 Key Distribution

Once a node gets a new key as a result of network merging, the key should be deployed within its previous network cell. There are several ways to achieve this:

- *Proactively* - each node receiving the key immediately forwards it to the others. This approach ensures prompt delivery of the key to all nodes, but it also generates a lot of unnecessary network traffic.
- *Reactively* - when a node receives a key, it does nothing. Only after detecting a message sent by a neighbor and signed with the old key, the node sends the new key further. This approach uses less resources, but it takes more time for the whole cell to get a stable key.
- *Combination* - the first node getting the new key (that is, the node which performed the merge) immediately forwards the key to its one-hop neighbors, since it knows that no other node in its previous cell has it yet. The other nodes do not distribute it right away, but rather when (if) they notice that a node still uses an old key. This approach keeps the number of necessary broadcast messages containing the key at a minimum.

In any of the given cases, the new key is encrypted using the old one before sending, giving all the other nodes the possibility to immediately start using it. The old key is saved for a short period of time, for possible latecomers. This can be done because in this particular case the key change was not performed explicitly for the purpose of preventing traffic analysis attacks.

In our implementation, described in Section 8.4.1, we use the *combination* approach.

8.3.4 Key Update

When created, each key has a companion key (called *update key*) used to periodically update it. The update key is never used on traffic that goes onto the network and therefore it is not prone to traffic-analysis attacks. The nodes must periodically update the main key. The new key can be computed using one-way hash functions such as SHA-1 [25] or MD5 [35], ensuring backward secrecy in the case the key gets broken at some stage. In addition to the ID of the key used to sign it, a message contains also the update-number saying how many times the key on the sender-node has been updated. That way, the receiver can easily compute the new key if it notices a mismatch, which could happen since we can't expect all the nodes to perform the update at exactly the same time. The local update will not take place if the received message has an invalid signature.

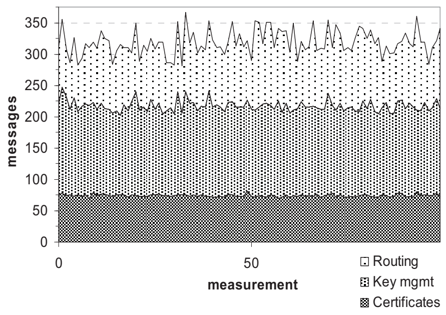


Figure 8.2: Traffic analysis of the first, non-optimized protocol implementation

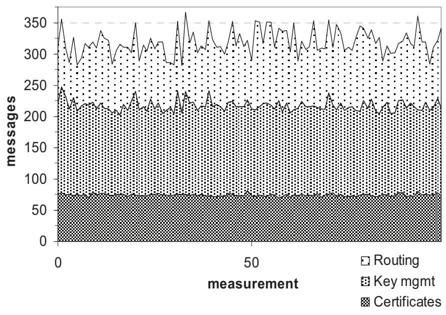


Figure 8.3: Results for the same scenario, after introducing the batching process

8.3.5 Exclusion of Nodes

Once authenticated, a node is a fully trusted member of the network. This poses the evident problem of how to exclude such a node once the device has been lost or, even worse, stolen by a malicious third party. At the present, exclusion of already authenticated nodes is not solved in SKiMPy and is part of ongoing and future work. Here, we describe some ideas on measures to be taken in order to ensure that such a node stays out of the network.

First, the node's certificate must be revoked, preventing the node from re-authenticating later at some stage. Since there is no central authority, a decision is reached on which node or person can perform the task of revoking certificates. If the certificates contain also additional attributes such as rank or role of the persons (assuming that the certificates do in fact represent persons, not devices), it can be decided that only certain roles/ranks (such as *leader*) can perform revocation and blacklisting. In theory, the leaders' devices might also be stolen, but in practice they should normally be physically well protected. It is important to ensure that the compromised node itself does not revoke and blacklist legitimate ones or, even worse, the whole network.

Next, the node's IP address should be put on a common blacklist. Assuming that IP addresses are bound to the certificates (as presented in e.g. [32]), the nodes would be unable to change their IP address. However, relying on fixed IP addresses might introduce new issues and should be considered carefully. Traffic coming from blacklisted nodes must be discarded at the lowest possible layer and, in case legally signed traffic coming from a blacklisted node is detected, the compromised key must be removed.

Additional methods might be used to ensure that devices cannot be used by unauthorized persons. One such example is a system relying on short range wireless authentication tokens. A token is installed into the personnel's vests or watches, ensuring confidentiality of the data and denying unauthorized access to the devices when they get out of their token's range [18].

8.3.6 Batching

To save resources as much as possible, our protocol makes the nodes learn about their neighborhood before acting, reducing the number of performed authentications and thus reducing directly CPU and bandwidth consumption. This is possible due to the fact that all nodes directly trust the same certificate authority and, therefore, if a node has been successfully authenticated before and has received the shared secret, we implicitly trust it.

Emphasis has been put on optimization with regards to number of messages sent out in the air. We measured the number of certificates and key management messages exchanged, and compared these figures to the number of routing messages needed from the moment when the nodes were turned on, up to the moment when a stable shared key was established. To perform these measurements, we used a static, wired test bed with 16 nodes.

Figures 8.2 and 8.3 show that introducing neighborhood awareness approximately halved the total number of messages and, proportionally, the time needed to reach a stable state. Moreover, the number of messages carrying certificates, whose size is much larger than other key management messages, has been reduced to approximately 23% of the initial number. The authentication was considered to be done after the exchange of certificates. Therefore, the results shown here are only an approximation, and might be slightly different when an actual authentication algorithm is used.

8.3.7 Additional Issues

The protocol's goal is to establish a secure network infrastructure. SKiMPy makes it impossible for a misbehaving node to induce a key that has either expired, or that would not have been selected in a normal operation. Such keys will be immediately discarded.

Timeouts are used during the *Key Exchange* phase (explained in Section 8.2) to ensure that a node does not end up in indefinite wait states or deadlocks as a result of possible link failures. Care must be taken for possible Denial-of-Service attacks in any of these cases.

In the *closing* phase of the rescue operation [33], the keys must be removed to prevent them from being possibly reused afterwards on a different rescue site.

8.4 Protocol Implementation and Evaluation

8.4.1 Implementation

Optimized Link State Routing Protocol (OLSR) [17] is a proactive routing protocol for ad-hoc networks which is one of the candidates to be used in our solution for the emergency and rescue operations. The olsr.org OLSR daemon [38] is the implementation we decided to test, since it is portable and expandable by means of

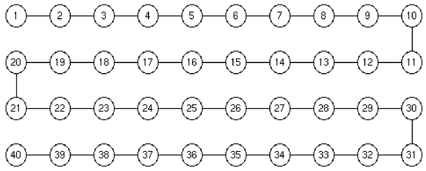


Figure 8.4: Example of a chain scenario

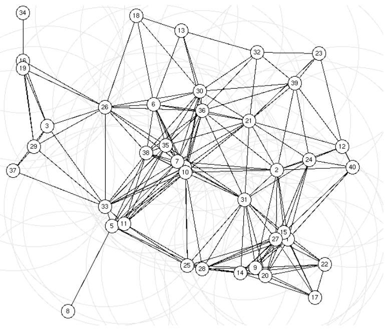


Figure 8.5: Example of a mesh scenario

loadable plugins. One example of such a plugin, present in the main distribution, is the Secure OLSR plugin [26]. The plugin is used to add signature messages to OLSR traffic, only allowing nodes that possess the correct shared (pre-installed) key to be part of the OLSR routing domain. One important functionality this plugin lacks is a key management protocol. Even though SKiMPy is mainly designed to protect all traffic and not only routing, it is still a good opportunity to test and analyze it in a realistic environment with a real routing protocol.

The key management protocol has been coded directly into the security plugin, although the plans are to make it as a separate one. X.509 certificates [28] and OpenSSL [37] are currently used to perform node authentication.

8.4.2 Evaluation Results

To facilitate development of this and other protocols, we created an emulation test bed, called NEMAN [34]. Routing daemons run independently, each attached to a different virtual Ethernet device. We use the monitoring channel of the emulator to analyze the keys used by each of the routing daemons. In order to test performance and scalability the protocol, we have made measurements from 2 to 100 nodes, with two very different kinds of scenario: chain and mesh. Figures 8.4 and 8.5 show example screenshots taken from the GUI, representing the two different scenarios.

In a chain scenario, the nodes are lined up in a single chain and the distance between all nodes in the chain is such that only the direct neighbors can communicate in a single hop with each other. We consider this to be the worst case scenario still giving full network connectivity. Given that all the nodes have to perform authentication with both their neighbors, this leaves no place for optimization, i.e. batching during the waiting period.

In a mesh scenario, however, nodes have multiple, randomly scattered neighbors, as it is natural in ad-hoc networks. Having multiple neighbors allows the protocol to exploit the batching phase, reducing traffic and resource consumption.

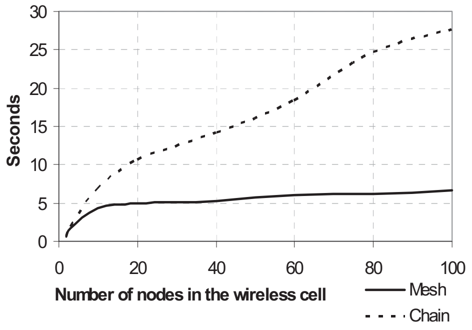


Figure 8.6: Time needed to achieve a stable shared key

Ten independent runs were performed for each number of nodes and each scenario. All the nodes were started simultaneously (which we assume is the worst case for our protocol), with a random key and key ID. To be able to meaningfully compare the results, the nodes were static and the density was constant. The delay in the batching period was set to be 1 second, i.e. half of the interval used by OLSR to send HELLO messages.

One important fact that the results on Figure 8.6 immediately show is that the protocol scales linearly with linear increase of the number of nodes and physical network area accordingly (thus giving the same density of nodes). After approximately 10 nodes, the total time became almost independent on the network size. By the fourth second, most authentications have already been performed and the key distribution process came into place. In some additional measurements, we introduced node movement using the random waypoint mobility model. As long as all of the nodes remained reachable and the density was constant, movement did not induce a notable delay.

We also proved that having multiple neighbors does in fact lower the time necessary to reach a stable state. This scenario gives less deviation as well, which is understandable since in the case of chain there is more fluctuation of keys, nicely seen in the GUI.

8.5 Related Work

Different authentication schemes are available as a starting point for key management.

Devices can exchange a secret or pre-authentication data through a physical contact or directed infrared link between them [3, 26]. Another way is for the users to compare strings displayed on their devices (a representation of their public key, distance between them, etc. as presented in [19]). Since user interaction in a rescue operation should be kept as minimum, we need a different approach.

Threshold cryptography schemes, such as [30] and [41] require all nodes that are going to perform signatures to carry a share of the group private key. The full signature is acquired by a certain, predefined number of nodes who present partial signatures computed using their shares. These schemes allow a small number of nodes to be compromised and still not to present a threat for the network. However, since we do not know the number of nodes that can be expected at the rescue scene and small partitions might always be present, this approach is not suited for our scenario.

Čapkun et al. [20] present a fully self-organized public-key management system that does not rely on trusted authorities, developed mainly for networks where users can join and leave without any centralized control. This is not applicable to networks used in rescue operations, where only authorized nodes are allowed to participate. In [21], they present a solution similar to ours, explained in Section 8.3.1, allowing nodes to authenticate each other by means of pre-installed certificates with a common authority. The advantages of such a system are twofold: first, the data in the network is more secure. Second, establishing trust and agreeing on a shared key is much more efficient, i.e., faster and less resources are consumed.

Related key management protocols can be roughly divided into the following three categories [16].

The first one relies on a fixed infrastructure and servers that are always reachable. Since we never know where accidents will happen, we should expect them to happen at places where we cannot rely on the fact that fixed infrastructure will be present.

The next category comprises contributory key agreement protocols, which are not suited for our scenario for several reasons. Such protocols ([1, 5, 12, 29, 30], to name a few) are based on Diffie-Hellman two-party key exchange [23] where all the nodes give their contribution to the final shared key, causing re-keying every time a new node joins or an existing node leaves the group. In an emergency and rescue operation, we can expect nodes to pop up and disappear all the time, often causing network partitioning and merging. Therefore, using contributory protocols would cause a lot of computational and bandwidth costs which cannot be afforded. Besides, most of these protocols rely on some kind of hierarchy (chain, binary tree, etc.) and a group manager to deploy and maintain shared keys. In a highly dynamic scenario this approach would be quite ineffective. Another reason why such protocols are not suited for us, is that in order for the nodes to be able to exchange keys, a fully working routing infrastructure has to be established prior to that. Since the routing protocol is one of the main things we need to protect, this is a major drawback. Asokan and Ginzboorg [12] present a password-based authenticated key exchange system. A weak password is known to every member and it is used by each of them to compute a part of the final shared key. This approach shares some already mentioned drawbacks and introduces new ones which conflict with our scenario and requirements. User interaction is needed and it is assumed that all the members are present when creating the key.

The last category are protocols based on key pre-distribution. The main characteristic of such protocols is that a pair or group of nodes can compute a shared key out of pre-distributed sets of keys present on each node. These sets of keys are either given by a trusted entity before the nodes come to the scene [4, 14, 21], or chosen and managed by the nodes themselves, as it is done in DKPS [16].

SKiMPy is different in the sense that it uses pre-installed certificates to perform direct authentication between two nodes. This makes it more simple and efficient.

8.6 Conclusion

In this paper, we presented a simple and efficient key management protocol, called SKiMPy, developed and optimized especially for highly dynamic ad-hoc networks. The protocol relies on the fact that there will be an *a priori* phase of rescue and emergency operations, within which certificates will be deployed on rescue personnel's devices. Pre-installed certificates are necessary due to the fact that highly sensitive data may be exchanged between the rescue personnel. The certificates make it possible for the nodes to authenticate each other without need for a third party present on the scene.

We described a proof-of-concept implementation, as well as evaluation results. The results show that SKiMPy performs very well and it scales linearly with the number of nodes. As part of further work we will analyze more in-depth different key selection and distribution schemes, authentication protocols, and fine tune certain protocol parameters, like the delays described in Section 8.2. Open issues like exclusion of compromised nodes, duplicate key ID numbers, denial of service attacks, etc. are also subject of further investigation.

Acknowledgment

This work has been funded by the Norwegian Research Council in the IKT-2010 Program, Project Nr. 152929/431. It has been also partly supported by the European Union under the E-Next SATIN-EDRF project.

References

[11] Alves-Foss, J., "An Efficient Secure Authenticated Group Key Exchange Algorithm for Large And Dynamic Groups", Proceedings of the 23rd National Information Systems Security Conference, pages 254-266, October 2000

[12] Asokan, N., Ginzboorg, P., "Key Agreement in Ad Hoc Networks", Computer Communications, 23:1627-1637, 2000

[13] Balfanz, D, Smetters, D. K., Stewart, P, Wong, H. C., "Talking To Strangers: Authentication in Ad-Hoc Wireless Networks", Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS'02), San Diego, California, February 2002

[14] Blom, R., "An Optimal Class of Symmetric Key Generation System", Advances in Cryptology - Eurocrypt'84, LNCS vol. 209, p. 335-338, 1985

[15] Bresson, E., Chevassut, O., Pointcheval, D., "Provably Authenticated Group Diffie-Hellman Key Exchange - The Dynamic Case (Extended Abstract)", Advances in Cryptology - Proceedings of AsiaCrypt 2001, pages 290-309. LNCS, Vol. 2248, 2001

[16] Chan, Aldar C-F., "Distributed Symmetric Key Management for Mobile Ad hoc Networks", IEEE Infocom 2004, Hong Kong, March 2004

[17] Clausen T., Jacquet P., "Optimized Link State Routing Protocol (OLSR)", RFC 3626, October 2003

[18] Corner, Mark D., Noble, Brian D., "Zero-Interaction Authentication", at The 8th Annual International Conference on Mobile Computing and Networking (MobiCom'02), Atlanta, Georgia, September 2002

[19] Čagalj, M., Čapkun, S., Hubaux, J.-P., "Key agreement in peer-to-peer wireless networks", to appear in Proceedings of the IEEE (Specials Issue on Security and Cryptography), 2005

[20] Čapkun, S., Buttyán, L., Hubaux, J.-P., "Self-Organized Public-Key Management for Mobile Ad Hoc Networks", IEEE Transactions on Mobile Computing, Vol. 2, No. 1, January-March 2003

[21] Čapkun, S., Hubaux, J.-P., Buttyán, L., "Mobility Helps Security in Ad Hoc Networks", In Proceedings of the 4th ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03), Annapolis, Maryland, June 2003

[22] Di Pietro, R., Mancini, L., Jajodia, S., "Efficient and Secure Keys Management for Wireless Mobile Communications", Proceedings of the second ACM international workshop on Principles of mobile computing, pages 66-73, ACM Press, 2002

[23] Diffie, W., Hellman, M., "New directions in cryptography", IEEE Transactions on Information Theory, 22(6):644-652, November 1976

[24] Eschenauer L., Gligor, Virgil D., "A Key-Management Scheme for Distributed Sensor Networks", Proceedings of the 9th ACM Conference on Computer and Communication Security (CCS'02), Washington D.C., November 2002

[25] Federal Information Processing Standard, Publication 180-1. Secure Hash Standard (SHA-1), April 1995

[26] Hafslund A., Tønnesen A., Rotvik J. B., Andersson J., Kure Ø., "Secure Extension to the OLSR protocol", OLSR Interop Workshop, San Diego, August 2004

[27] Hollick, M., Schmitt, J., Seipl, C., Steinmetz, R., "On the Effect of Node Misbehavior in Ad Hoc Networks", Proceedings of IEEE International Conference on Communications, ICC'04, Paris, France, volume 6, pages 3759-3763. IEEE, June 2004

[28] Housley, R., Ford, W., Polk, W. and D. Solo, "Internet X.509 Public Key Infrastructure", RFC 2459, January 1999

[29] IEEE, "IEEE Std. 802.11b-1999 (R2003)", http://standards.ieee.org/getieee802/download/802.11b-1999.pdf

[30] Luo, H., Kong, J., Zerfos, P., Lu, S., Zhang, L., "URSA: Ubiquitous and Robust Access Control for Mobile Ad-Hoc Networks", IEEE/ACM Transactions on Networking, October 2004

[31] Matsumoto, T., Imai, H., "On the key predistribution systems: A practical solution to the key distribution problem", Advances in Cryptology - Crypto'87, LNCS vol. 293, p. 185-193, 1988

[32] Montenegro, G., Castelluccia, C., "Statistically Unique and Cryptographically Verifiable (SUCV) Identifiers and Addresses", NDSS'02, February 2002

[33] Plagemann, T. *et al.*, "Middleware Services for Information Sharing in Mobile Ad-Hoc Networks - Challenges and Approach", Workshop on Challenges of Mobility, IFIP TC6 World Computer Congress, Toulouse, France, August 2004

[34] Pužar, M., Plagemann, T., "NEMAN: A Network Emulator for Mobile Ad-Hoc Networks", Proceedings of the 8th International Conference on Telecommunications (ConTEL 2005), Zagreb, Croatia, June 2005

[35] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992

[36] Stajano, R., Anderson, R., "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks", 7th International Workshop on Security Protocols, Cambridge, UK, 1999

[37] The OpenSSL project, http://www.openssl.org/

[38] Tønnesen A., "Implementing and extending the Optimized Link State Routing protocol", http://www.olsr.org/, August 2004

[39] Wallner, D., Harder, E., Agee, R., "Key management for Multicast: issues and architecture", RFC 2627, June 1999

[40] Wong, C., Gouda, M. and S. Lam, "Secure Group Communications Using Key Graphs", Technical Report TR 97-23, Department of Computer Sciences, The University of Texas at Austin, November 1998

[41] Zhou, L., Haas, Z., "Securing Ad Hoc networks", IEEE Network, 13(6):24-30, 1999

Chapter 9

NEMAN: A Network Emulator for Mobile Ad-Hoc Networks

Authors: Matija Pužar, Thomas Plagemann

Affiliations: Department of Informatics, University of Oslo
{matija, plageman}@ifi.uio.no

Publication: The 8th International Conference on Telecommunications (ConTEL 2005), Zagreb, Croatia, June 2005

Abstract: Development of applications and protocols for wireless ad-hoc networks has always been a challenge. Specific characteristics such as frequent topology changes due to nodes moving around, popping up or being turned off, need to be considered from the earliest stages of development. Since testing and evaluation using genuine wireless devices is both expensive and highly impractical, other tools need to be used in the development phase. Simulators give a very detailed model of lower layers' behaviors, but code often needs to be completely rewritten in order to be used on actual physical devices. Emulators present a trade-off between real test beds and simulators, providing a virtual wireless network at the lowest layers, and yet allowing real code to be run on the higher layers. In this paper, we present such an emulation platform, called NEMAN, that allows us to run a virtual wireless network of hundreds of nodes on a single end-user machine. NEMAN has shown to be an important and very useful tool during development of different applications and protocols for our project, including a key management protocol and a distributed event notification service.

9.1 Introduction

Information sharing is a mission critical element in rescue and emergency operations. Mobile ad-hoc networks (MANETs) could provide a useful infrastructure to support information sharing, but appropriate applications are needed. In addition, middleware support has to be present to facilitate efficient application development for this type of infrastructure. In the Ad-Hoc InfoWare project [9], we are addressing these needs by developing middleware services for information sharing. The core building blocks of these services are knowledge management, a local and a distributed event notification service, resource management, and security and privacy management. As part of the development process, it is necessary to analyze and compare design and implementation alternatives for the building blocks, understand qualitatively and quantitatively the design trade-offs, to test whether the protocols and algorithms actually work, and to evaluate their efficiency and compare them with related solutions from others. Basically, there are three kinds of development environments that support these tasks: simulation, emulation, as well as implementation and field tests.

The development environment that comes closest to real live deployment are field tests of MANETs. However, field tests of wireless scenarios are expensive with respect to the number of devices and persons needed to perform them as well as the time it takes to prepare them. Furthermore, it is very hard to entirely control all parameters in a field test and to perform repeatable experiments. Field tests are clearly not perfectly suited to support the development of middleware protocols, services and applications for MANETs. On the other hand, both simulation and emulation provide controlled environments which enable repeatable experiments and are generally much cheaper than field tests. To facilitate the development and testing of such applications and protocols, it is important to carefully choose a suitable simulation or emulation tool.

Simulators, such as GloMoSim [15] and ns-2 [12], have long been used in the ad-hoc field and make it possible to experience very diverse communication situations and a large scale of deployment. One of the goals of these simulators is to give a detailed representation of the physical layer. A major drawback is that learning how to use and program a simulator like ns-2 takes a substantial amount of time. Furthermore, the code that has been developed for these simulators needs to be rewritten for later deployment on real systems. By using an emulating platform, however, real processes run in real time and one can immediately start writing the very same code that will be later used on wireless devices.

In particular, we have the following requirements on a simulation or emulation environment for the development of protocols, middleware services and applications in the Ad-Hoc InfoWare project:

- *Minimal initial effort*: installing and learning to use a particular simulation or emulation environment should not consume too much time and effort, so that it can also be efficiently used from novices in shorter projects, like master theses.

- *Costs*: we need an affordable solution, with regards to hardware and software costs, as well as human resources. Thus, the environment should be able to run on a single standard PC.
- *Scalability*: the chosen platform should be able to run a high number of nodes without severe performance loss.
- *Portability*: the code developed for the applications and protocols should be portable to genuine wireless devices with minor or no changes at all.
- *Realistic network layer:* our protocols are supposed to utilize existing ad-hoc routing protocols, therefore, a real routing protocol should run in the simulation or emulation environment. During the development process, our main concern is connectivity and loss of connectivity between nodes. Quality of service and specific lower layer issues such as collisions in the air, hidden terminals, etc. are of lower importance for us. We mainly need to reflect the effects of mobility in MANETs on connectivity.
- *Possible comparability*: much work in the area is being done using ns-2 and we want to be able to compare our solutions with those from others without re-implementing them in our simulation or emulation environment. Using standard formats for scenario files, such as those from ns-2, would enable us to perform experiments with the same scenarios and to compare our results with results obtained with other tools.

Many researchers are aware of the need for appropriate development environments of MANET protocols, but the majority is being focused on link layer and network layer issues. Therefore, we could not find existing simulation or emulation environments that completely fulfill our requirements. Inspired by the approach of the network emulator MobiEmu [16], we have developed an emulation platform called NEMAN. To the best of our knowledge, NEMAN is the only platform able to emulate MANETs consisting of hundreds of nodes on a single PC. Based on scenario descriptions from ns-2, NEMAN controls the physical connectivity between the virtual mobile nodes. Currently, we use the Open Link State Routing Protocol (OLSR) [1] with NEMAN to establish and maintain the IP layer of the emulated network. The processes in these virtual mobile nodes bind to virtual network interfaces, i.e., TAP interfaces, available in the Linux kernel. By this, the code developed for NEMAN can be used in real wireless nodes with a minimal effort. It is the aim of this paper to describe the design, implementation and first evaluation of NEMAN, and to analyze its strength and weaknesses.

The rest of the paper is structured as follows. Section 9.2 shows some of the most important related network emulators. In Section 9.3, we present the architecture of NEMAN. Implementation details are described in Section 9.4, while in the following sections we explain the application development process (Section 9.5) and describe our experiences in using NEMAN for the development of a key management protocols and a distributed event notification service in the Ad-Hoc InfoWare project (Section 9.6). Performance and scalability are described in Section 9.7. Finally, Section 9.8 gives a conclusion and ideas for the future work.

9.2 Related Work

A common way to perform emulations is to have a single machine for each emulated node. In such cases, filtering at the MAC layer is used to achieve the notion of wireless topology, often by means of *iptables*.

MobiEmu consists of several *slave* nodes, as well as one *master* node. As the master gives instructions on topology changes, the slaves set local *iptables*-rules preventing them from hearing traffic from those nodes they have no physical connectivity with. However, the concept of having separate physical or virtual machines for each emulated node, made MobiEmu very impractical for us to use it as such.

MNE (Mobile Network Emulator [6]) uses a static network infrastructure to interconnect devices. Each device has two interfaces, where one acts as a mobile emulation control channel while the other is used for the emulated wireless network. The latter can be an actual wireless interface, allowing for some lower layer effects (such as collisions) to be taken into account as well. Information about topology changes is sent through the control channel, causing the nodes to set or remove *iptables*-rules accordingly, as it is done in MobiEmu. The main problem of this approach is that it still needs a separate device for each emulated wireless host.

EMWIN [17] improves the issue with the number of physical machines by allowing each node to have several network interfaces, each acting as a separate wireless node. EMWIN intends to provide emulation of some MAC layer effects by introducing an additional emulated MAC (eMAC) layer. Again, due to a relatively high number of machines required, this approach is still impractical for our needs.

MobiNet [7] consists of *core* nodes, used to emulate topology-specific and hop-by-hop network characteristics, and *edge* nodes. It is able to emulate a much larger number of virtual wireless devices by having multiple Virtual Edge Nodes (VNs), with different IP addresses, on each *edge* node. MobiNet has a built-in routing protocol (DSR) and emulates MAC layer effects as well. Although the number of physical devices required to run MobiNet is drastically reduced, and the platform seems to be very well developed, its setup is still somehow complicated, with regards to our requirements.

JEmu [3] was developed by the Networks & Telecommunications Research Group (NTRG) to emulate the radio components of their particular communication stack. To represent nodes in the emulation, JEmu uses genuine wireless de-

Table 9.1: Properties of Various Emulators for Mobile Networks

	MobiEmu	MNE	EMWIN	MobiNet	JEmu
Usage	✓				
Low costs				✓	
Scalability			✓	✓	
Portability	✓	✓	✓	✓	✓
Network layer	✓	✓	✓	✓	✓
Comparability				✓	

vices with different types of wireless communication links, as well as stationary machines. JEmu has a somehow different approach when it comes to topology simulation. Every packet is first sent to the emulation engine which then decides whether certain nodes are able to receive it or whether there should be a collision, in which case it depends on the specific configuration what should be done.

Table 9.1 summarizes roughly the properties of these related works with respect to our particular requirements. As it can be seen from the table, none of them fulfills all the requirements.

A different platform, designed for wired networks, is IMUNES [14]. It provides virtualization of the complete network stack functionality, allowing for simultaneous operation of multiple independent network stack instances, i.e. virtual nodes, within a single FreeBSD kernel. In contrast, NEMAN is implemented in user space and uses a single network stack of the Linux kernel. Another difference is that IMUNES provides no support for mobile scenarios.

9.3 Architecture

NEMAN is designed to emulate a relatively large scale wireless network, consisting of up to hundreds of nodes, within a single physical machine. With that respect, NEMAN is closest to MobiNet. The NEMAN architecture comprises the following three elements, as shown on Figure 9.1:

- the *user processes* represent actual applications and protocols that are being tested, including routing daemons,
- the *topology manager* manages virtual network interfaces and performs packet switching according to the topology information at a certain moment in time, and
- the *graphical user interface* (GUI), used to visualize the emulated network

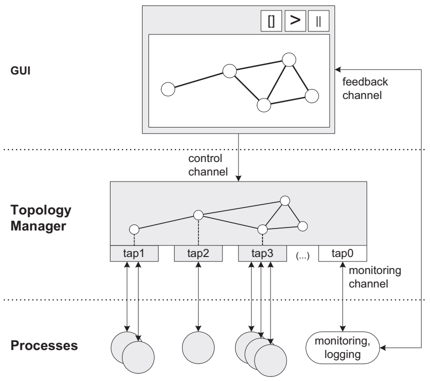


Figure 9.1: NEMAN architecture

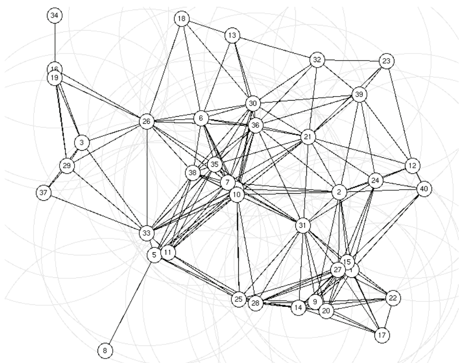


Figure 9.2: Example screenshot taken from the GUI

and to induce the topology information to the topology manager

All the components, including the topology manager, run in the user space of the Linux operating system. Root-privilege is needed to configure the virtual network interfaces (standard *ifconfig* command) and to be able to use the required socket option SO_BINDTODEVICE.

User processes communicate through this basic network infrastructure by hooking to virtual Ethernet network devices, called TAP devices. TAP devices are available in the Linux kernel and provide low level support for Ethernet tunneling. User processes can send and receive data via TAP interfaces using the classical socket API, thus achieving portability of code. The only requirement for the sockets is to use the specific socket option SO_BINDTODEVICE. This is an important requirement as it ensures that a process' socket will listen and send only to the specified interface, and thus not interfere with traffic addressed to other process running on the same machine. By this, we introduce the notion of *virtual nodes*, comprising all processes hooked to a certain TAP interface.

NEMAN by itself operates on the link layer, with minor interventions to the network layer as well, such as internal hop-by-hop routing. This is necessary because it is not possible for multiple routing daemons to correctly use the same kernel's routing table (as explained in Section 9.4). Routing daemons are an example of standard user level processes that are hooked to the TAP interfaces. They are needed to provide a working IP infrastructure to all other user level processes that are communicating via IP based protocols through the TAP interfaces. Thus, routing daemons are an important prerequisite to implement and test middleware and application layer protocols, which was exactly our goal.

The *topology manager* is the core of NEMAN. It is the user-space application creating and maintaining the TAP devices. Since TAP devices provide Ethernet tunneling, we ensured the possibility of running any network layer protocol on top. Every frame received on a TAP interface is available to the topology manager, and every frame forwarded by the topology manager to a TAP interface is available to the processes hooked to it. In other words, when the topology manager gets a frame sent to one of its TAP interfaces, it can then decide to forward it to some of the other interfaces (or none), according to the topology information it has at the moment. One TAP interface (in our case, tap0) is reserved as the *monitoring channel*, having an open bidirectional connection to all the other TAP interfaces, independent from the topology. This is a very important feature, allowing us to perform analysis of the network traffic using standard tools such as *tcpdump* or *ethereal*. Moreover, having in mind that the monitoring channel works both ways, we are able to use the same channel to induce traffic into the virtual network from the "outside world". This feature comes useful when applications or services need to be triggered at a specific moment of time. An example of such a service is shown in Section 9.6.2.

The implementation of the GUI is currently based on MobiEmu's GUI. It is a Tcl/Tk script, independent from the topology manager and can run on a separate machine. The GUI shows the current position of nodes, their transmission ranges and links between nodes that can directly communicate with each other (Figure 9.2). Topology and node movement data are acquired from standard ns-2 scenario files, created by, for example, ns-2's *setdest* program. Scenario files are interpreted sequentially, allowing us to introduce some application-specific events at specific moments in the emulation, thus achieving repeatable results. Information about topology changes is sent to the topology manager through the control channel, in form of UDP packets. The GUI allows also any user process to give it some feedback, so that important state changes in a user process can be visualized as e.g. color changes in the GUI. An example, where this has proved to be a very useful feature, is described in Section 9.6.1.

The current implementation of the emulator provides us a working network infrastructure, essential for the development process of higher layer applications. In the conclusion, we discuss some possible future extensions, including the emulation of characteristics typical for wireless networks, such as collisions in the air, hidden terminals, obstacles, etc.

9.4 Implementation Details

One of the major problems we noticed was that the Linux kernel gracefully ignores all incoming packets that have been sent from one of its own interfaces. Although in most situations this is a reasonable thing to do, with respect to security issues and prevention from possible message loops, it was not an option in our case, where local interfaces were the only ones communicating between themselves. The solution was to implement the send-to-self (STS) patch developed by Ben Greear, which fixed the problem, allowing for local traffic to be received as well.

The next problem emerged when the first ARP packets started coming from our test-applications. For any ARP packet coming to the machine, independent on the IP address being queried, all local interfaces that hear the query answer with their own MAC address. Again, this might be a useful feature in most of the standard situations where an IP address represents a physical machine, but in our case the amount of unnecessary traffic generated (with false information) was unacceptable. There are two possibilities to solve this problem. The first one is to implement a kernel patch, called *hidden*, developed by Julian Anastasov. This patch allows for certain devices to be hidden from other devices (hence the name), when it comes to ARP requests. The second possibility is to let topology manager directly answer all ARP requests between its TAP interfaces. Although the second approach might reduce a bit on realism at the lower layers, for our case, where we are mainly concerned about connectivity and topology changes, it is acceptable. Furthermore, it turned out that the multi-hop routing problem can be solved in a corresponding way.

The multi-hop routing problem is caused by the fact that IP addresses of "remote" virtual nodes are tied to local TAP interfaces. This contradiction prevents the routing protocols to set routes towards such addresses, which is the case for all our emulated nodes. Therefore, the use of the kernel's routing table is impossible in our case. Using dynamically created *iptables*-rules to perform routing on a single machine proved not to work either. We decided to solve the problem on a higher layer, without introducing additional patches into the kernel. The implementation of the OLSR routing protocol we are currently using in NEMAN, the *olsr.org OLSR daemon* [13], writes every route change to the standard output. This enables us to induce that information directly into the topology manager, through a parser developed especially for that purpose. The parser serves as a bridge between the routing daemons and the topology manager, translating their outputs to control messages. The topology manager keeps an internal routing table for each virtual node, which is then used for hop-by-hop forwarding of packets. In the network, this forwarding appears as if every intermediate node had retransmitted the packet until its final destination. By solving the routing issue in the described way, no changes were needed to the routing daemon and the kernel. Depending on an implementation, this approach might allow us to use other routing protocols as well, with either no changes at all or just minor changes to the routing daemon's output and/or to our parser. Reactive routing protocols, such as AODV [8], present a case where some code must reside in the kernel. This is necessary since such protocols need to intercept and hold outgoing packets used to trigger route discovery. Porting such code to NEMAN might prove to be much more difficult than in the case of OLSR.

9.5 Application Development

One of the main points of using an emulator is to be able to port applications from the emulating platform to genuine wireless devices without need having to change

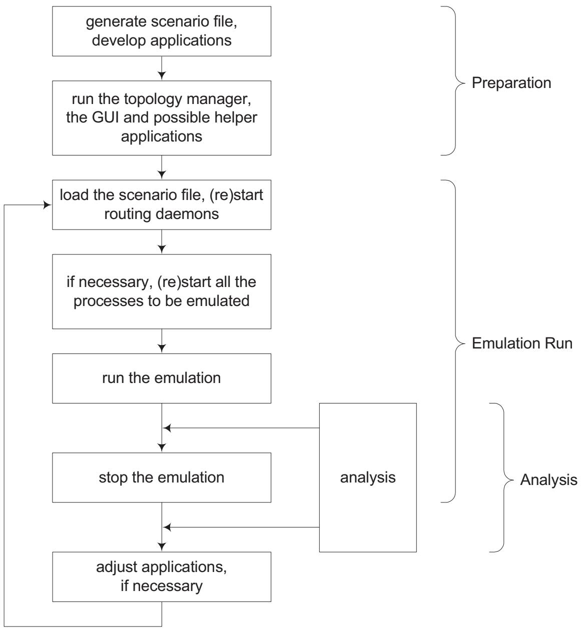

generate scenario file, develop applications
run the topology manager, the GUI and possible helper applications
Preparation
load the scenario file, (re)start routing daemons
if necessary, (re)start all the processes to be emulated
run the emulation
Emulation Run
stop the emulation
analysis
Analysis
adjust applications, if necessary

Figure 9.3: Workflow diagram of using NEMAN

the code. Any application able to use e.g. Ethernet (eth*), 802.11 (wlan*) or similar devices can be used directly with TAP devices as well. The only precondition is that, in order not to mix with other applications' data, an application has to listen and send only to the specified devices, which is accomplished by using the previously mentioned SO_BINDTODEVICE option when creating the socket.

Figure 9.3 shows the typical process of using NEMAN. The process is roughly divided into three phases. In the preparation phase, the scenario files and initial user process' code are developed, and the emulator core is initiated. In the next phase, user processes are started. The analysis phase can overlap with the run-phase, since traffic can be monitored on the fly through the monitoring channel. The last two phases are then iterated until the wanted functionality is achieved.

Apart from the applications being tested, additional helper applications might be needed to, for example, restart routing daemons on demand or to listen to messages triggered by custom events in the scenario file. The monitoring channel provides the means to perform analysis on what is going on in the network on the fly, by hooking to tap0 with e.g. standard tools like tcpdump. In addition, all the traffic can be saved for a later, more detailed analysis.

9.6 Experiences

NEMAN was developed primarily to test and evaluate applications and protocols for the Ad-Hoc InfoWare project. Here, we present our experiences with some of the applications and protocols we tested and which already benefit from the emulation platform.

The olsr.org OLSR daemon was the first example showing that already existing applications might need no modifications to be able to work with NEMAN. Indeed, at each moment the output from a certain daemon, showing its 1-hop and 2-hops neighbors, would fully match what was displayed in the GUI.

9.6.1 A Simple Key Management Protocol for MANETs

The SKiMPy key management protocol [10] is used to establish a symmetric shared key between the rescue personnel's devices. The key provides the means for establishing a secure network infrastructure between authorized nodes, while keeping out unauthorized ones. SKiMPy is designed and optimized for highly dynamic ad-hoc networks and it is completely autonomous, requiring no user interaction at all. In the current implementation, the key management protocol has been coded directly into the security plugin [5] of the OLSR routing daemon.
The emulator was an essential tool to test the performance and scalability of SKiMPy. We tested it for two different static scenarios, namely chain and mesh.

In a chain scenario, all nodes are lined up in a single chain and the distance between the nodes in the chain is such that only the direct neighbors can communicate in a single hop with each other. We consider this to be the worst case scenario for SKiMPy, since its performance benefits from having more neighbors.

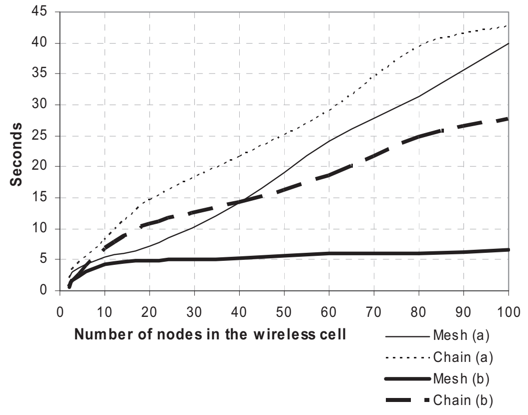


Figure 9.4: Time needed to achieve a stable shared key

In a mesh scenario, nodes have multiple, randomly scattered neighbors, as it is natural in ad-hoc networks. Having multiple neighbors allows the protocol to reduce traffic and resource consumption.

We have measured the time needed for the whole network to agree on a shared key. Two different platforms have been used, a Pentium 4 2.4GHz running Linux 2.4.21 (a) and a dual Xeon 2.80GHz running Linux 2.6.10 (b).

Ten independent runs were performed for each number of nodes and each scenario. The results on Figure 9.4 show that the key management protocol scales linearly with linear increase of the number of nodes and physical network area accordingly (thus giving the same density of nodes). It can also be seen that in the first case the machine presented a bottleneck with regards to the ability to deliver packets in real time (see discussion in Section 9.7). In the second case, the total time became almost independent on the network size with networks consisting of 10 or more nodes.

One of the useful features of MobiEmu's original GUI is that it accepts certain feedback messages. A special application constantly monitors all traffic on the monitoring channel (tap0) and analyzes signed OLSR packets, containing the ID number of the key used to perform the signature. When a change is noticed, the ID number is converted to a 24-bit RGB color code and sent as feedback to the GUI, which then colors the node on the screen accordingly. That way, we got a simple and yet effective way to see in real time how the protocol works and how the keys are spread through the network.

To conclude, NEMAN has played an important role in the process of developing, testing and evaluating SKiMPy. It presented us both numerical and graphical proofs that the protocol indeed worked as expected.

9.6.2 Distributed Event Notification Service (DENS)

DENS [11] is a communication tool in our architecture, providing an asynchronous communication mechanism using the publish/subscribe model which is well suited for a highly dynamic environment such as our scenario. Some of the nodes in the network are chosen as so called DENS-nodes. These nodes have the role of mediators or brokers between the subscribers and publishers, so both subscriptions and notifications are sent to them. In a MANET, disconnections causing partitioning are frequent and the DENS-nodes may therefore get disconnected as well. We want the service to be highly available so our solution should survive such network partitions. This is achieved by replicating information about subscriptions among the DENS-nodes. However, the replication of state information together with network partitions can easily lead to inconsistent replicas. One important tasks for DENS-nodes is therefore to regain a consistent state as fast as possible. Scalability is also an issue since having too many DENS-nodes would create a lot of traffic when synchronizing and replicating information about subscriptions and notifications, so we need to run tests to find the right trade-off between availability of the service and scalability.

The first implementation of the DENS protocol is currently under development and NEMAN has proven to be an essential tool for analyzing and debugging the protocol. To trigger DENS-nodes to perform subscriptions and notifications, which in the real scenario will be done at the application layer, we use the monitoring channel for its other purpose, i.e. inducing messages. Control messages are sent through the monitoring channel to DENS nodes either directly from the shell, or from the GUI. Special lines can be added to the scenario file, causing the GUI to forward them at the specified time, through a proxy application, into the monitoring channel.

9.7 Performance and Scalability

We have already seen in Section 9.6.1 that the hardware configuration on which NEMAN is running can present a bottleneck. This bottleneck can have impact on the accuracy of time related results, because NEMAN is no longer able to deliver packets in real time. There are several factors that need to be taken into the performance analysis, including CPU time slicing, packet queues, data copying when forwarding packets, context switching, control messages from the GUI and the routing parser, quantity of traffic in the network, other external processes running, etc. All of these factors play an important role in the performance degradation which is inevitable having many virtual nodes. A detailed analysis is part of ongoing and future work.

One of the main places for performance improvement is the GUI. Although it is easy to use and to be extended with new functionalities, its ability to accurately visualize the movements in the scenario is reduced as the number of nodes and active links is increased, even with the minimal smoothness factor. To avoid this problem, the graphical display can be temporarily turned off in cases where results are needed quickly and they do not depend on time. This can be done either by closing the application or by simply minimizing the window. Another problem with large scenarios (either with regards to the time domain or number of nodes) is that the GUI loads the whole scenario in memory before starting to interpret it. Assuming that reading the scenario file on the fly would not impose a new bottleneck, this might be a point to investigate.

9.8 Conclusion and Future Work

In this paper, we described the requirements, design, implementation, and experiences with our emulation platform for MANETs, called NEMAN. Development of middleware services for emergency and rescue operations is the main focus of Ad-Hoc InfoWare project. In this context, NEMAN has already proven to be an important and very useful tool for the development of our key management protocol SKiMPy and the distributed event notification service.

There are several other simulation and emulation environments of MANETs, however, to the best of our knowledge, NEMAN is the only emulation platform allowing to perform MANET emulations of up to hundreds of nodes on a single machine. By this, NEMAN is a cheap and efficient environment to develop middleware protocols, services, and applications. Furthermore, the code that has been developed on NEMAN can be easily ported and deployed on genuine wireless devices. Obviously, this facilitates considerably the step from development in an emulation environment to “real life” field trials and deployment. We hope that this will also contribute in the future to increase the number of research results that are not only developed and evaluated within a simulation or emulation environment, but are also tested with real devices in field trials.

While the current implementation of NEMAN has already proven to be very useful, there are still some aspects of NEMAN we would like to improve, respectively to extend. We envisage three possible threads of future work. First, we are interested to increase the number of nodes that can be efficiently emulated. To overcome the resource limitations of a single PC, we are looking into design and implementing a NEMAN version that can run concurrently on multiple PCs. We might even include heterogeneous computing platforms into the simulation to emulate the software where it actually should be running. For creating such a distributed simulation for ad-hoc networks, we are investigating distributed simulation, component, and multi-agent architectures and technologies such as the High Level Architecture (HLA) [2] and Foundation for Intelligent Physical Agents (FIPA) architecture [4] for multi-agent systems.

Second, we would like to extend the functionality of the topology manager to not only emulate the connectivity between nodes according to a scenario description, but also to emulate some other link layer properties, like Quality-of-Service, hidden terminals, etc. The emulation of these aspects will require much more resources than just deciding whether a virtual device should receive a packet or not. Thus, the availability of a distributed NEMAN version might be necessary to perform such emulations with larger number of nodes.

The third thread of possible future work is concerned with the GUI. The GUI is currently the performance bottleneck in NEMAN with regards to the graphical visualization of the mobility. This bottleneck is caused by the fact that the GUI is implemented in Tcl/Tk. A re-implementation with a language that can be compiled should lead to substantial performance improvements. Furthermore, we are interested to extend the functionality of the GUI. We would like to use the GUI to turn on and off nodes at any stage of the emulation by just clicking on them with, be able to create new nodes on the fly, and even influence the link layer properties by clicking on them.

Acknowledgments

This work has been performed in the Ad-Hoc InfoWare project and we would like to thank all project team members and related colleagues for many fruitful discussions and other contributions. Special thanks go to Katrine S. Skjelsvik for contributing with valuable feedback on NEMAN as a patient beta tester, and Erek Göktürk for providing important insights into related work. Furthermore, we would like to acknowledge the MobiEmu project, whose GUI we used as a starting point for creating NEMAN.

References

[1] Clausen T., Jacquet P., “Optimized Link State Routing Protocol (OLSR)”, RFC 3626, 2003.

[2] IEEE-SA Standards Board, IEEE Standard 1516

[3] Flynn, J., Tewari, H., O’Mahony, D., “A Real Time Emulation System for Mobile Ad Hoc Networks”, Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference, 2002.

[4] Foundation for Intelligent Physical Agents (FIPA) standard no: SC00001L, FIPA Abstract Architecture Specification, 2002.

[5] Hafslund A., Tønnesen A., Rotvik J. B., Andersson J., Kure Ø., “Secure Extension to the OLSR protocol”, OLSR Interop Workshop, San Diego, August 2004.

[6] Macker, J. P., Chao, W., Weston, J. W., “A low-cost, IP-based mobile network emulator (MNE)”, MILCOM 2003 - IEEE Military Communications Conference, 2003, 22, 481-486.

[7] Mahadevan, P., Rodriguez, A., Becker, D., Vahdat, A., “MobiNet: A Scalable Emulation Infrastructure for Ad Hoc and Wireless Networks”, UCSD Tech. Report CS2004-0792, 2004.

[8] Perkins, C., Belding-Royer, E., “Ad hoc On-Demand Distance Vector (AODV) Routing”, RFC 3561, July 2003.

[9] Plagemann, T., *et al.*, “Middleware Services for Information Sharing in Mobile Ad-Hoc Networks - Challenges and Approach”, Workshop on Challenges of Mobility, IFIP TC6 World Computer Congress, Toulouse, France, August 2004.

[10] Pužar, M., Andersson, J., Plagemann, T., Roudier, Y., “SKiMPy: A Simple Key Management Protocol for MANETs in Emergency and Rescue Operations”, Technical Report #319, Department of Informatics, University of Oslo, February 2005.

[11] Skjelsvik, K. S., Goebel, V., Plagemann, T., “A Highly Available Distributed Event Notification Service for Mobile Ad-hoc Networks”, ACM/IFIP/USENIX 5th International Middleware Conference (Middleware 2004), Toronto, Canada, October 2004.

[12] The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns/

[13] Tønnesen A., “Implementing and extending the Optimized Link State Routing protocol”, http://www.olsr.org/, August 2004.

[14] Zec, M., Mikuc, M., “Operating System Support for Integrated Network Emulation in IMUNES”, Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure / ASPLOS-XI, Boston, October 2004.

[15] Zeng, X., Bagrodia, R., Gerla, M., “GloMoSim: a Library for the Parallel Network Simulation Environment”, Proceedings of the 12th Workshop on Parallel and Distributed Systems, 1998.

[16] Zhang, Y., Li, W., “An Integrated Environment for Testing Mobile Ad-Hoc Networks”, MOBIHOC’02, EPFL Lausanne, Switzerland, 2002.

[17] Zheng, P., Ni, L. M., “EMWIN: Emulating a Mobile Wireless Network using a Wired Network”, 5th ACM international workshop on Wireless mobile multimedia, Atlanta, Georgia, 2002

Chapter 10

Cross-layer Overlay Synchronization in Sparse MANETs

Authors: Thomas Plagemann, Katrine Stemland Skjelsvik, Matija Pužar, Aslak Johannessen, Ovidiu Valentin Drugan, Vera Goebel, Ellen Munthe-Kaas

Affiliations: Department of Informatics, University of Oslo
{plageman, katrins, matija,
aslakjo, ovidiu, goebel, ellenmk}@ifi.uio.no

Publication: The 5th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2008), Washington DC, USA, May 2008

Abstract: Mobile Ad-Hoc Networks maintain information in the routing table about reachable nodes. In emergency and rescue operations, human groups play an important role. This is visible at the network level as independent network partitions which are for some time stable before their members change through merging or partitioning. We use the information from stable routing tables to optimize the synchronization of Mediators in a Distributed Event Notification System. In a stable partition each node has the same information, thus a single Mediator can efficiently coordinate the synchronization, while all other Mediators just receive updates. We show in our experiments that just a few seconds are needed until routing tables stabilize and all nodes have a common view of the partition. We present a heuristic to determine the proper time to synchronize. Furthermore, we show how exceptions, like disappearing coordinating Mediators and unexpected messages, can be efficiently handled.

10.1 Introduction

In order to efficiently handle crises and emergencies, emergency and rescue (ER) teams benefit from well working communication infrastructures for command, control and coordination. However, first responders are typically confronted with an environment where no communication infrastructure is available, either because it was nonexisting, or the earlier existing ones have been destroyed. Therefore, wireless Mobile Ad-Hoc Networks (MANETs) formed by devices carried by ER personnel are often the only means to establish a communication infrastructure. However, the mobility of the ER personnel combined with the size of the emergency area (which is typically multiple times larger than the coverage of individual IEEE 802.11 radios) and obstacles in the area reflecting radio waves, leads to the situation that there is often not one single MANET connecting all ER personnel. Instead, multiple partitions might exist and change over time through merging and partitioning. Typically, these partitions correspond to groups of ER personnel that have a common task to fulfill. Due to the dynamics of ER operations, groups might need to change their locations, and group memberships might change. This is reflected at the network level through changes in the routing table. Evidence for such group mobility is not only given by our study of ER operations, but also confirmed by recent studies of social mobility [2] and community detection [6] in opportunistic networks.

We have developed a Distributed Event Notification Service (DENS) [10] to support remote patient monitoring. DENS uses Mediators to replicate subscriptions, to enable graceful degradation in case of network partitions, and to convey subscriptions and notifications from source to destination. If there is connectivity to the destination the Mediator uses the OLSR MANET routing protocol [4] and IP to transport the packets to their destination. However, if a destination node is turned off or in a different partition, OLSR and any other MANET routing protocol fails. Therefore, the Mediators form an overlay on top of the MANET to perform delay tolerant transport through so-called “store-carry-forward” operations [10]. The replication of undelivered subscriptions and notifications increases the probability that one of the Mediators at a later point in time can join a partition with formerly unreachable destinations. The number of Mediators can be dynamically adapted to the particular ER operation and by this trade availability of DENS against resource consumption, i.e., the more Mediators the higher the availability and the higher the resource consumption.

The message ferry approach [12] is determining the mobility, including speed and trajectory of special nodes called ferries, to make sure that a previously unreachable destination and the ferry are coming into communication range. This is not in general possible in ER operations. Therefore, we replicate undelivered subscriptions and notifications to all Mediators, similar to the approach in epidemic routing [11] where packets are forwarded to neighbors and then spreads through the network, hence the name. However, we make use of a proactive routing protocol and do not depend on the event that two nodes meet, to enable exchange of undelivered messages. Instead, we can synchronize Mediators immediately after they join a common network partition, which are typically formed by different ER teams.

There exist various works related to routing in intermittently connected networks, or sparse MANETs. In [13] Zhang gives an overview of different approaches for the store-carry-forward routing. They differ in how they select the next-hop-node, e.g. random selection, using knowledge about the whereabouts of nodes, or predict future location. Other approaches assume some knowledge such as last known location of the destination, and therefore only forward packets in the same area. There exist some cross-layer approaches such as EMMA [9] where synchronous communication is used if possible; in case of no end-to-end connection, epidemic routing is used instead.

Synchronization of Mediators seems to be simple, but several complicating factors have to be considered: First, each node has its own view of "its" network partition which does not necessarily correspond to the view of the other nodes in this partition. Second, merging of partitions is not an atomic action, the routing daemon in each node discovers new nodes iteratively over several time steps. Third, nodes are mobile and there might never be a globally correct definition whether partition merging has finished or not. Fourth, propagation of messages from source to destination through epidemic routing might take quite a long time depending on the network topology and the movement of the nodes; however, certain messages in ER applications have stringent timing requirements, like an alarm from a patient monitoring application. Finally, bandwidth is a scarce resource and synchronization among multiple Mediators should be as efficient as possible.

We propose in this paper to leverage the existing information about the network topology from the OLSR routing protocol. By this each Mediator can establish its own view of the network without putting any additional load onto the network. Furthermore, we use the fact that ER operations are performed in groups, resulting in partitions that are stable for some time (with respect to the nodes that form the partition) before new mergings or partitionings occur. We show through simulation studies that the time it takes to merge network partitions and the time it takes until all nodes in the new partition have updated routing tables, is rather short. By assuming a common view among the Mediators in each partition, we can for each partition immediately identify a coordinator that acts on behalf of all Mediators in its partition. This in turn enables us to minimize the number of exchanged messages in the synchronization process.

In the remainder of this paper, we briefly describe DENS in Section 10.2. In Section 10.3, we analyze how useful information from the IP layer, i.e., the OLSR routing table is. In Section 10.4, we outline the basic idea of the synchronization protocol and describe how exceptions are handled, followed by some conclusions in Section 10.5.

10.2 DENS

In publish-subscribe systems, subscribers express their interest in events through subscriptions, and publishers publish events of interest. DENS is designed to support information exchange even in the presence of network partitions. Subscription information is therefore replicated in the network. There is a trade-off between offering a reliable service, but at the same time not saturate the network by replicating too much information. Filtering of events is therefore performed as close to the source as possible. DENS supports different subscription languages, both simple subject-based languages, and content-based languages where the subscriber can specify the content of the notification it wants to receive information on, e.g., a value range of health sensor data.

DENS has the following components: (1) Subscriber, (2) Publisher and (3) Mediator. Subscriber and Publisher run in every node, and the Mediator runs on some of the nodes. The number of Mediators is dependent on the network size, density and dynamicity. The Subscriber Component sends subscriptions to a Mediator running on the same or another node. The Mediator parses received subscriptions to find keywords that are used to locate potential publisher nodes. The subscriptions are then sent to these publisher nodes. The Publisher filters events according to the subscriptions. When an event of interest occurs, it sends a notification to a Mediator. The notifications are matched with the subscriptions, and then delivered to interested subscribers. Thus, Mediators decouple Subscribers and Publishers.

DENS keeps track of Mediators by listening to beacons sent by such nodes in its partition. The Mediators form a DENS overlay. In addition to forwarding subscriptions and notifications, Mediators replicate subscriptions and possibly notifications to obtain a higher degree of reliability in the presence of partitionings caused by disconnections or that a node is turned off. The task of the overlay formed by the Mediators is to enhance reliability and support delay-tolerant routing of subscriptions and notifications in case of partitions.

Subscribers and publishers send their subscriptions and notifications to a Mediator in their own partition using end-to-end paths set by the network routing protocol. This means that the Mediator is an indirection. Delivering subscriptions and notifications, and replicating subscriptions and undelivered notifications, are done by using the underlying routing protocol and the synchronization protocol. The synchronization protocol is initiated when there are new nodes in the partition. The presence of new nodes provides the means for delivering undelivered stored subscriptions and notifications to the newly connected nodes, and replicating subscriptions and undelivered notifications to newly arrived Mediators. Because of the network partitionings, the Mediators can have an inconsistent view of subscriptions. In the next section, we describe how we can use information from the routing table to detect topology changes and initiate the synchronization process among Mediators.

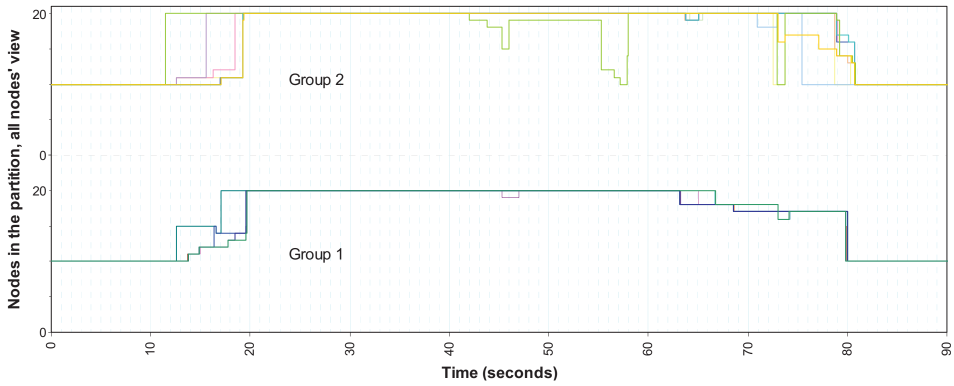


Figure 10.1: Two mobile groups merging and partitioning (each subgraph represents the view of one group)

10.3 Routing Table Information

One important key element to enable efficient design of middleware protocols over Sparse MANETs, is information about nodes that can be reached through a multi-hop route at a given point in time and the prediction of future connectivity. Parts of this information might be gathered from external sources, like GPS satellites or base stations, but we aim to design our solutions such that they work also when no external information is available. The other possibility to gather this information is that the middleware components periodically broadcast messages, like in Hypergossiping [7], to detect partition mergings, etc. Since bandwidth is a scarce resource, we aim to minimize the number of broadcast messages.

In order to gather at the middleware layer information about network partitions, mergings, and partition membership information in a non-intrusive manner, we leverage the information that is already available at the network layer in the routing table. In our previous studies [5] we have observed that the routing protocol holds updated information about the neighborhood of a node if the node is actively involved in communication. This claim holds for both proactive and reactive routing protocols. However, proactive routing protocols maintain topology information also if there is no (or not sufficient) communication. The proactive routing protocol OLSR periodically sends beacons (*HELLO messages*) to inform other nodes of its presence. In addition, OLSR tries to maintain at each node a consistent view of the network by exchanging topology information with the other nodes in the network. Whenever there is a change in the topology, the routing table is recalculated. Each entry in the routing table contains information on the destination node, the next hop node, the estimated number of hops to the destination, and the interface used for communication.

In order to optimize the synchronization of Mediators after partition mergings, we need to understand the dynamics of both the merging process and the partitioning process and how it is reflected in the routing tables of the individual nodes. We have performed a number of experiments with the emulation tool NEMAN [8] to analyze how often the local routing table changes over time, whether the change frequency allows us to deduce that a merging or partitioning has finished, and how long it takes until all nodes in one partition have the same view of their partition. We instrumented the code of the OLSR daemon to extract and log all changes of membership information, which enables us to identify how many neighbors each node's routing protocol reported at any point in time. The OLSR's interval for sending HELLO messages is set to 1 second in all experiments.

In order to verify our hypothesis that groups which move in larger areas result in stable membership information for a substantial amount of time, we performed experiments with non-static groups, moving according to the reference point group mobility pattern. The nodes were moving at 2 units/s in an area of 600x600 units. Figure 10.1 illustrates a representative case in which two groups of 10 nodes came in contact approximately after 11 seconds and remained in contact for approximately 77 seconds. For each node there is one line in the graph that shows how many partition members this node has registered. Overlapping lines indicate a consistent view among multiple nodes. The figure shows on each group subgraph a single line for most of the time, meaning that both groups have a stable view of the network. Occasionally, due to the mobility of nodes a few nodes have a different view.

Table 10.1 shows the results for a selected set of experiments, including two static groups of 1, 10, or 20 nodes each. Merging and partitioning events were introduced artificially by creating or removing contact between the two groups at a certain number of merging points. The merging time and partitioning time were measured on a global basis, i.e. from the moment the first node noticed the change to the moment when all the nodes in the partition had the same view.

Table 10.1: Resulting times for merging and partitioning

Topology	Merging points	Groups	Merg. time	Part. time
Chain	1	20+20	10,97s	8,73s
Mesh	1	20+20	8,47s	6,79s
Mesh	5	20+20	7,40s	7,80s
Full mesh	full mesh	20+1	0,28s	2,41s
Full mesh	full mesh	10+10	1,17s	1,32s

In addition to experiments with group mobility models and especially designed topologies, we have also performed experiments with the random waypoint model as a worst case analysis. We have simulated results for 20 nodes in areas of size 500x500, 1000x1000 and 1500x1500, and 250 units radio range. As expected membership changes in routing tables for very dense networks are rare, i.e., membership does not change during the entire run. In larger areas there are also longer periods in which the individual routing tables do not change. In the studied case where the area size was 1000x1000, this was often more than 10 seconds, while in larger areas this stable period was often several times longer. When a merging takes place, the routing daemons recalculate old routes and add new routes towards the new nodes. This takes approximately 5 seconds on each node. This is dependent on the location of the node and the number of new nodes.

10.4 Synchronization Protocol

The basic idea for the synchronization protocol is to use information from the routing table to initiate synchronizing of data after a merging of two or more partitions. The protocol is initiated in a node when it detects a routing protocol change that indicates a partition merge, but only after it assumes that the routing table has stabilized. During the synchronization process some of the Mediators resume specialized roles as *partition_representatives*. A *partition_representative* is responsible for synchronizing data in its old partition. The role of being a *partition_representative* for an old partition is taken by the Mediator with the highest node ID. Since we assume that each Mediator keeps a record of all other Mediators in its partition, the *partition_representative*'s identity is implicitly known without any message exchange. Among the *partition_representatives* one takes the role as *coordinator*. The *coordinator* has the responsibility of coordinating the synchronization process among the *partition_representatives*. The *coordinator* is the *partition_representative* having the highest node ID. After the *partition_representatives* have synchronized data among themselves, they send updates to the Mediators of their old partitions.

We first describe the heuristic used to determine when the synchronization should be initiated, before we explain the basic protocol without exception handling and what kind of conditions we assume. Then we describe how exceptions are handled if these conditions do not hold.

10.4.1 Synchronization Initialization

The heuristic to determine when to initiate the synchronization uses two timestamps and three threshold values:

- Timestamp tstart records the time a new node is registered in the routing table after a stable period.
- Timestamp tlast records the time the last change of membership information in the routing table was detected.
- Threshold value Ts is an estimate whether the routing table is stable, i.e., there are no membership changes during the period [tlast, tlast + Ts].
- Threshold value TGV estimates the time it takes for all nodes in a partition to have the same membership information after the last membership change in the local table was detected.
- Threshold value TE is used to ensure that the heuristic is able to start the synchronization process from time to time even if there is never a stable routing table.

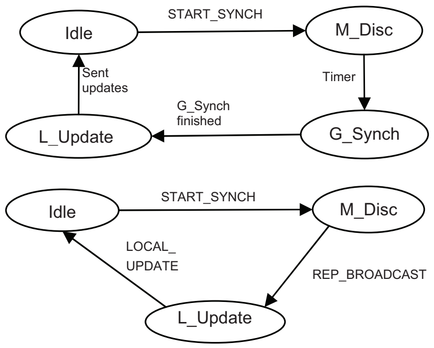


Figure 10.2: Mediator and *partition_representative* states

The heuristic is started when a new node is registered in the routing table. Both t_{start} and t_{last} are assigned the current time. Each time a change of the membership information occurs, t_{last} is updated with the current time. Normally, the synchronization process is started if the routing table is stable and all nodes have the same membership information. If the exception occurs that the routing table changes continuously for a too long time, synchronization is enforced even if the routing tables are not stable. The pseudo code of the heuristic is given below.

```
$t_{start}$, $t_{last}$ := $t_{current}$;
repeat {
   if (membership_change) {$t_{last}$:=$t_{current}$;}
until ($t_{last}$ + max($T_S$,$T_{GV}$) < $t_{current}$ || $t_{start}$ + $T_E$ < $t_{last}$) }
Start_Synch;
```

Based on our experiments, we are currently using 5 seconds for T_S and T_{GV}. T_E has to be adapted to the application requirements to balance between resource consumption and availability.

10.4.2 Basic Protocol

For each node the protocol has three phases: the Mediator Discovery phase, where Mediators from merging partitions are discovered and one Mediator from each old partition takes the role of a *partition_representative*; the Global Synchronization phase where the *coordinator* is selected and the *partition_representatives* exchange information; and the Local Update phase where the *partition_representatives* send updates to the Mediators in their old partition. In each phase timers are set to ensure that the phase always completes. The events triggering the different phases are shown in Figure 10.2.

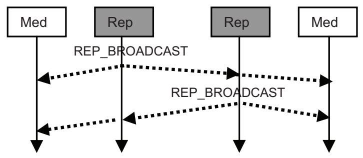


Figure 10.3: Mediator Discovery (messages)

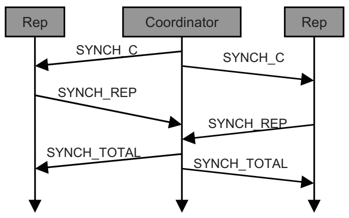


Figure 10.4: Global Synchronization (messages)

The Basic Protocol runs under these assumptions:

- each node knows about every other node in the new (merged) partition,
- each Mediator knows about every other Mediator in its old partition,
- all Mediators in an old partition are synchronized, and
- during the synchronization process no new nodes arrive, no nodes disappear, and no new subscriptions or notifications are sent.

In the following we describe the phases, roles, and messages.

Mediator Discovery

A Mediator enters this phase when it receives a START_SYNCH message. The Mediator starts a timer. Each Mediator examines its set of known Mediators and decides whether it is a *partition_representative*. The Mediators taking the role of a *partition_representative,* floods a REP_BROADCAST message. This message includes a list of the Mediators it represents. When a Mediator receives a REP_BROADCAST from its own *partition_representative*, it enters the Local Update phase and cancels the timer. The *partition_representative* listens for REP_BROADCASTs from the other partitions and waits until there is a timeout. It then enters the Global Synchronization phase.

Global Synchronization

All Mediators that enter this phase are *partition_representatives,* in addition one of them takes the role of being a *coordinator.* Again, this role is taken by the Mediator having the highest node ID. The messages used in this phase are: SYNCH_C, SYNCH_REP, and SYNCH_TOTAL. A timer is started when the Mediators enter the phase.

The *coordinator* sends SYNCH_C containing a summary of its subscriptions to the other *partition_representatives*. The other *partition_representatives* compare the summary with their own content and reply with the message SYNCH_REP containing data the *coordinator* is lacking, in addition to a summary of its own data. When the *coordinator* has received replies from all the *partition_representatives*, it sends SYNCH_TOTAL updates to the *partition_representatives*, i.e., its own subscriptions in addition to subscriptions received from the other *partition_representatives*. The *coordinator* cancels its timer,

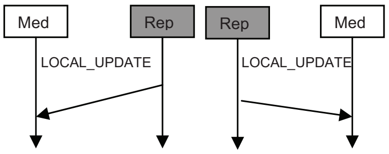


Figure 10.5: Local Update (messages)

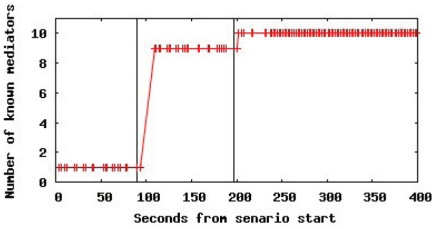


Figure 10.6: Nr. of Mediators from one Mediator's perspective

resumes status as an ordinary *partition_representative*, and enters the Local Update phase. When the *partition_representatives* receive the SYNCH_TOTAL message from the *coordinator*, they cancel the timer and enter the Local Update phase.

Local Update

In the last step of the protocol, the *partition_representatives* send LOCAL_UPDATE messages to the Mediators in their old partition. This includes information about subscriptions, but also about new Mediators. Each ordinary Mediator in this phase starts a timer, then it awaits the arrival of a LOCAL_UPDATE message from its *partition_representative*. When the LOCAL_UPDATE message arrives, the timer is cancelled and the Mediator resumes ordinary activity.

We have implemented the basic protocol. Figure 10.6 shows the number of known Mediators from one Mediator's perspective when testing it in a scenario where there are two merging events; one at 90 seconds and the second merging at 195 seconds. The increasing number of known Mediators shows that the nodes in the network have detected a partition merging, the routing tables are stabilized, and the Mediator Discovery phase is started.

10.4.3 Exception Handling

We now discuss how exceptions are handled in the different phases. Examples of exceptions are that the Mediators do not have the same view of the partition membership, that Mediators may appear or disappear during the synchronization process, and that Mediators in the old partition may not be fully synchronized when the synchronization protocol starts. It is important to notice that we cannot assume at any stage that the nodes have the exact same view of where they are in the synchronization process, and what the members of a partition are. The protocol therefore needs to be robust enough to manage these situations. In the following, we discuss the different phases of the protocol from one Mediator's point of view. The notable exceptions are shown in the Tables 2-5. The handling of the exceptions is dependent on the phase and the role of the Mediator.

Table 10.2. Exception Handling when Idle

Role	Exception	Handling
M	REP_BROADCAST	Start synchronization

Table 10.3. Exception Handling in Mediator Discovery phase

Role	Exception	Handling
$M + R$	START_SYNCH	Stack request
	LOCAL_UPDATE	Receive data
R	timeout without any received REP_BROADCAST	Proceed to L_Update
M	REP_BROADCAST from an unexpected R in its old partition	Reconsider the identity of R for its old partition
	timeout, no received REP_BROADCASTs from nodes in its old partition	Reconsider role to R

Table 10.4. Exception Handling in the Global Synch phase

Role	Exception	Handling
$C+ R$	START_SYNCH	Stack request
	REP_BROADCAST	Stack request
	LOCAL_UPDATE	Receive data
C	timeout without having received any SYNCH_REPs	Proceed to L_Update
	timeout but has only received some of the expected SYNCH_REPs	Proceed with reduced set of recipient Rs
R	timeout without having received SYNCH_C	Reconsider role to C
	timeout without having received SYNCH_TOTAL	Proceed to L_Update
	SYNCH_C from wrong C	Respond with SYNCH_REP but continue to wait for SYNCH_C from true C

Table 10.5. Exception Handling in the Local Update phase

Role	Exception	Handling
$M +R$	START_SYNCH	Stack request
	REP_BROADCAST	Stack request
M	timeout, has not received LOCAL_UPDATE	Proceed

A Mediator enters the Mediator Discovery phase when it receives a START_SYNCH or a REP_BROADCAST message. It then starts a timer. If a Mediator receives a new START_SYNCH message during this phase, it will just stack the request and enter the Mediator Discovery phase again after it has finished its current synchronization process. If it receives a LOCAL_UPDATE message out of order, it receives data that can be handled locally immediately. It may be the case that the Mediators in the old partition do not have exactly the same view of the partition membership, so a Mediator can receive a REP_BROADCAST from a node that it is aware of but did not consider being the *partition_representative*. In this case, the Mediator reports to the new *partition_representative*. If the timer fires for a node that is assumed not to be a *partition_representative,* it will reconsider which Mediator should be *partition_representative*. If it is the next Mediator having the highest ID, it sends a REP_BROADCAST, if not it will restart the timer. If the timer fires for the *partition_representative*, it sees if it has received any REP_BROADCAST messages, if not, it goes directly to Local Update phase.

In the Global Synchronization phase only *partition_representatives* participate, and one of them takes the *coordinator* role. In this phase both START_SYNCH messages and REP_BROADCAST messages are stacked and handled when the process is finished. LOCAL_UPDATE messages are just received but not handled. If the *coordinator* disappears, the remaining *partition_representatives* will at timeout reconsider their roles, and the one with the next highest id becomes the new *coordinator*. If all *partition_representatives* but the *coordinator* disappear, the *coordinator* will at timeout enter the Local Update phase. It may happen that none, two or more elect themselves as a *coordinator.* If none starts as a *coordinator*, then there will be a timeout where the *partition_representatives* reconsider their role. If there is a SYNCH_C from a non-assumed *coordinator*, the *partition_representatives* will respond to it but await a SYNCH_C from its true *coordinator before* proceeding to the Local Update phase.

As in the previous phase, both START_SYNCH messages and REP_BROADCAST messages received during the Local Update phase are stacked and handled when the process is finished. If a timeout fires, this indicates that something went wrong, i.e., the *partition_representative* is gone.

If subscriptions or notifications are received by a *partition_representative* at any phase, it will send it as LOCAL_UPDATE messages. If a Mediator is not a *partition_representative* or a *coordinator,* it will replicate it to the other Mediators.

10.4.4 Complexity

In order to compare the complexity of our synchronization protocol with a simple gossiping style synchronization, we use a simple analytic model. We assume two merging network partitions with x respectively y mediators that are synchronized within their partition. We measure the complexity in terms of number of Mediator to Mediator update processes. In our synchronization protocol, first the *partition_representatives* update each other, and afterwards they update the Mediators in their old partition, leading to 1 + (x -1) + (y - 1) updates. If instead all Mediators exchange information with all other Mediators, in the best case each Mediator from partition 1 updates all Mediators from partition 2 or vice versa. This leads us to x*y updates. The complexity is therefore O(x + y) for our synchronization protocol and O(x*y) for gossiping style protocols. The synchronization protocol in addition is deterministic since all Mediators are updated after a merging, and not only when e.g. Mediators become direct neighbors and initiate synchronization

10.5 Conclusions

One fundamental decision for the design of DENS is to use the proactive MANET routing protocol OLSR at the IP layer and to establish an overlay of Mediators that (1) increase availability of DENS through replication, and (2) perform delay tolerant transport of destinations in different partitions. Beside the advantage that a standardized routing protocol can be used to forward messages to the destinations if there is a route, we use the routing table information to optimize the Mediator synchronization in the overlay. OLSR maintains continuously at each node membership and topology information about its partition. Changes in the membership indicate a network merging or partitioning. If the set of member nodes in a partition is for some time unchanged, all nodes in the partition have the same view. Through simulations we have demonstrated that this assumption is correct, and we have quantified for different scenarios how long it takes until partitions are stable and all nodes have the same membership information. By observing the routing table, we can identify partition mergings and estimate when the merging has finished without exchanging any additional messages. The fact that all nodes have the same membership information, enables us to optimize the synchronization of Mediators since a single Mediator can act on behalf of the others in its partition. Since all nodes in a partition are known, the “election” of a representative is based on the node ID and no messages need to be sent. Additionally, all non-representative Mediators act as hot-standbys in case the representative disappears unexpectedly.

Different optimizations can be done to improve the efficiency of the protocol. These include to prevent too frequent synchronizing among Mediators in case of frequent topology changes; to use a compact representation of the subscriptions in the summaries sent in the Global Synchronization phase; and to use information about disappeared nodes from the Mediator’s local routing table. To prevent Mediators from repeatedly synchronizing with the same Mediators, the Mediators can remember when and with which Mediators they have synchronized. Bloom filters [1] can be used to summarize data. If a node detects disappeared nodes, some of the timers might be cancelled and the Mediators resume the protocol quicker, e.g., if a Mediator is waiting for a REP_BROADCAST from its assumed *partition_representative* and it detects that this node is gone, it can resume the protocol as if the timer has fired.

However, even without optimizations, the number of messages exchanged increases only linearly with the number of Mediators in the absence of exceptions. We argue that even in the worst case, our synchronization protocol does not perform worse than an approach based on Epidemic Routing in terms of bandwidth consumption and delivery delay since we are not depending on the fact that two nodes meet and we in most cases synchronize more than two nodes. We will continue to work on the implementation of the protocol and evaluate it with real world traces and compare its performance with Epidemic Routing.

Acknowledgments

This work was funded by the Norwegian Research Council in the IKT-2010 Program, Ad-Hoc InfoWare Project No. 152929/431, and supported by the NoE CONTENT.

References

[1] Bloom, B., Space/Time Trade-offs in Hash Coding with Allowable Errors, *Communication of ACM, 13(7)*, July 1970

[2] Boldrini, C., Conti, M., Passarella, A., Impact of Social Mobility on Routing Protocols for Opportunistic Networks, *1st IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC 2007)*, Helsinki, Finland, June 2007

[3] Camp, T., Boleng, J., Davies, V., A Survey of Mobility Models for Ad Hoc Network Research, *Wireless Communication and Mobile Computing (WCMD)*, 2002

[4] Clausen, T., Jacquet, P., Optimized Link State Routing Protocol (OLSR), *RPC 3626*, October 2003

[5] Drugan, O. V., Plagemann, T., Munthe-Kaas, E, Predicting Time Intervals for Resource Availability in MANETs, *The IEEE International Workshop on Ad Hoc and Ubiquitous Computing (AHUC2006), Taichung,* Taiwan, June, 2006

[6] Hui, P., Yoneki, E., Chan, S., Crowcroft, J., Distributed Community Detection in Delay Tolerant Networks, *ACM SIGCOMM Workshop (MOBIARCH)*, Kyoto, Japan, August 2007

[7] Khelil, A., Marrón, P.J., Becker, C., Rothermel, K., Hypergossiping: A General Broadcast Strategy for Mobile Ad Hoc Networks*, Ad hoc Networks Journal*, 2006

[8] Pužar, M., Plagemann,T., NEMAN: A Network Emulator for Mobile Ad-Hoc Networks, *8th Int. Conf. on Telecommunications (ConTEL)*, Zagreb, Croatia, June 2005

[9] Musolesi, M. Mascolo, C., Hailes, S., EMMA: Epidemic Messaging Middleware for Ad hoc Networks Personal and Ubiquitous Computing, Springer, vol. 10, no. 1, pp. 28-36, February 2006

[10] Skjelsvik, K. S., Goebel, V., Plagemann, T. A Highly Available Distributed Event Notification Service for Mobile Ad-hoc Networks, *ACM/IFIP/USENIX 5th International Middleware Conference (Middleware 2004),* October, 2004

[11] Vahdat, A., Becker, D., Epidemic Routing for Partially Connected Ad Hoc Networks, *Technical Report CS-2000-06, Department of Computer Science, Duke University*, 2000

[12] Zhao, W., Ammar, M. Zegura, E., A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks, *5th ACM Symp. on Mobile ad hoc networking and computing (MobiHoc)*, 2004

[13] Zhang, Z., Routing in Intermittently Connected Mobile Ad Hoc Networks and Delay Tolerant Networks - Overview and Challenges, *IEEE Communication Surveys and Tutorials*, January 2006

Chapter 11

Information Sharing in Mobile Ad-Hoc Networks: Metadata Management in the MIDAS Dataspace

Authors: Ellen Munthe-Kaas, Aslak Johannessen, Matija Pužar, Thomas Plagemann

Affiliations: Department of Informatics, University of Oslo
{ellenmk, aslakjo, matija, plageman}@ifi.uio.no

Publication: The 10th International Conference on Mobile Data Management: Systems, Services and Middleware, Taipei, Taiwan, May 2009

Abstract: An approach to information sharing in mobile adhoc networks (MANETs) is to store on every node a small amount of metadata describing what information resources exist in the network and where they reside, allowing applications to first search locally for information about suitable resources, and next request relevant information from a node that contains the resource. The characteristics of MANETs in general and sparse, delay-tolerant networks in particular, make the task of maintaining and disseminating metadata across all nodes difficult, particularly in the presence of scarce resources. We have designed and implemented three protocols which use different approaches to metadata dissemination: epidemic one-to-one routing, one-to-many broadcasting that utilises the characteristics of the shared radio medium of wireless networks, and a protocol where a group consisting of e.g. rescue team leaders is given priority in the dissemination process and afterwards serves as multiple starting points for further dissemination. In the MIDAS project we aim to produce middleware that speeds up MANET ap-

plication development. By prototyping the metadata management component of MIDAS and testing the prototype with each of the three protocols in the network emulator environment NEMAN, we have measured bandwidth usage and performance. The implemented broadcast protocol is measured to use substantially less bandwidth than epidemic routing. The group protocol shows that the group indeed gets priority; it is comparable to the broadcast protocol in terms of bandwidth usage. The broadcast protocol has been used successfully in field tests of the MIDAS middleware.

11.1 Introduction

Efficient information sharing is vital in rescue and emergency operations. Due to strict requirements in this domain we need information sharing services that work in environments where no communication infrastructure exists or where it has been destroyed. Mobile Ad-Hoc Networks (MANET) is a promising technology to be leveraged for generic information sharing services. However, in MANETs bandwidth is strictly limited by the physical characteristics of the shared radio medium. In our application domain we must expect frequent network partitions and remerges due to the possible low density and high mobility of the involved nodes. The number and type of devices on the site may change substantially over time during the incident, as rescue team members go to and fro. In addition the devices may have limited computing and storage capabilities; rescue teams cannot be expected to carry more than a modern mobile telephone and/or a hand-held device, e.g. a PDA. Still the devices should be able to set up and partake in the MANET. Since information sharing is vital, information sharing services in the MANET must survive short or long term network partitions and function properly also when networks (re)merge. On the other hand, the duration of such scenarios is fairly short, from a few hours to a few days.

When density in a MANET increases, the network degrades drastically due to an increase in packet collisions, thus in a realistic setup the MANET is limited to less than a hundred nodes. In the presence of heterogeneous storage capabilities and unpredictable and frequent network topology changes, one should place data replicas on well-chosen nodes to enhance data availability. The rest of the nodes then needs to know which nodes keep which data replicas. Such "yellow-page" information constitutes metadata about available resources.

Due to the small size of MANETs and the typically short duration of a scenario, the amount of globally shared metadata can be kept small, thus it is feasible to store all globally shared metadata information at all nodes. If the metadata is kept up to date, an application can query locally what kind of data items exists and where to find a replica. This does not require any communication and not much processing. When the application knows where to locate a resource, it can fetch or update the data item in question.

The focus of this paper is when and how to propagate metadata information, given that all nodes store all available globally shared metadata items locally and cooperate to disseminate metadata updates quickly enough to maintain a sufficiently up-to-date view of available resources. Metadata at each node must be kept consistent in the presence of metadata updates and network topology changes. Any solutions must take into account that bandwidth is a scarce commodity, and storage space and processing power likewise.

The work of this paper was performed in the context of the MIDAS project [1], in which we produce middleware that speeds up MANET application development. As part of the MIDAS prototype we have designed and implemented the Global Metadata Manager (GMDM) component whose task is metadata maintenance. We present three protocols for metadata propagation: (i) a basic epidemic routing protocol, (ii) a broadcast protocol using one-to-many communication, and (iii) a protocol where a group of nodes consisting of e.g. rescue team leaders is given first priority, and afterwards serves as multiple starting points for further dissemination. All three protocols are robust to network partitioning and merging. The MIDAS prototype with the GMDM component has been tested extensively with each of the three protocols in the network emulator environment NEMAN [2] to measure their bandwidth usage and performance. In addition proof-of-concept applications using the broadcast protocol have been successfully exposed to field tests under the 32nd international super prestige cyclocross race of Gieten in 2007, and the four day walking event in Nijmegen 2008.

The remainder of the paper is organised as follows: In Section 11.2 the relevant parts of the MIDAS architecture are presented. Relevant characteristics of MANETs and MANET routing techniques are reviewed in Section 11.3. In Section 11.4 we explain the metadata propagation protocols in detail. We describe the test setup and test results in Sections 11.5 and 11.6 respectively, and conclude in Section 11.7.

11.2 MIDAS Architecture

From the MIDAS architecture we primarily present the main ideas behind the MIDAS Data Space (MDS) component [3, 4], which is responsible for transparent information sharing. In MDS relational-style tables form the unit of shareable data. Ideally, if the same unit is accessed simultaneously by different applications using MDS, it is available for all and with identical contents. In practice however, data may be unavailable or users perceive different contents due to e.g. network disruption and propagation delays.

The main components of the MDS architecture are the following: A local storage stores the table replicas, while the Global MetaData Manager (GMDM) keeps track of on which nodes which replicas are stored. The original decision of where to store table replicas is made by the Data Allocator. The Data Synchroniser makes sure that modifications of a table are propagated to all replicas. The Query Analyser forwards application requests to the local storage or a remote node.

GMDM maintains information about tables and the location of their replicas. The functionality of GMDM bears some resemblance to the data dictionary in database systems, by its maintaining data about (shared) data, i.e., metadata. We need to keep the amount of metadata and the frequency of metadata updates at a reasonable level. The total amount of metadata is kept at a minimum by distinguishing between two kinds of metadata:

1. **Metadata detailed descriptions**: Each node that stores a table replica, maintains its full metadata description.
2. **Globally shared metadata**: For each table, the identities of remote nodes that store replicas of it.

In MANETs it is impossible to keep a complete global view of all table replicas at every node at all times, due to frequent network partitions and merges. Our design choice is still to let every node maintain locally as complete an overview as possible. That is, the local overview should comprise information about all table replicas residing in the node's own network partition (with a possible propagation delay), but obviously the local overview is not required to be the same on two nodes in different partitions unless and until the two partitions are merged. Thus, if an application requests information about a table, it can consult its local GMDM. If a replica is stored locally, the application can locally obtain the replica's full metadata description, query its contents, and perform updates. If not, GMDM can furnish the application with a list of nodes that store a replica. It is thus sufficient to keep at all nodes metadata of the second kind. A more sophisticated classification and utilisation of metadata can be found in [5].

The MDS component relies on the MIDAS Communication and Routing (CRT) component to provide network services. In MIDAS we have chosen OLSR [6] as our routing protocol for the case of mobile ad-hoc networks. OLSR maintains at each node information about the topology of the whole network. A main feature of OLSR is that it is proactive, i.e., it regularly exchanges topology information with the other nodes in the network; this is essential since the proper functioning of MDS requires the routing protocol to maintain at all times a fairly updated view of the topology.

There are two main situations where the globally shared metadata of a network partition may change:

- When a new table replica is created locally: In this case the Data Allocator informs GMDM.
- When two networks merge: Whenever CRT registers a topology change, it reports the change to MDS. If the change implies the arrival of a new neighbour node, GMDM contacts the new neighbour and exchanges globally shared metadata information with it.

In both cases GMDM should in a second phase cooperate with GMDM at other nodes in the network on propagating its new metadata items. Finally, in case new metadata is received from other nodes, either as a result of the encountering of new neighbours or as part of the metadata propagation phase, GMDM is responsible for doing local metadata updates.

In our application domain the size of the globally shared metadata will remain quite small. As a typical example, consider a MANET of 50 nodes. Each globally shared metadata item consists of a nodeid and a table name, say a total of 10+40 characters. A few (say 5) administrative tables will have replicas on all nodes. In addition there might be around 30 shared tables with an average of 5 replicas each. With a little bit of clever encoding this adds up to a few Kbytes of globally shared metadata per node.

Our main concern is how and when to perform the exchange and propagation of the globally shared metadata. In the rest of the paper we for short use the term "metadata" to mean what is named "globally shared metadata" in the above.

11.3 Routing in MANETs

Our main idea for the choice of metadata propagation protocols, is to reuse known techniques. For this purpose we have looked into routing protocols of dense MANETs and routing in delay-tolerant networks. Before presenting our design choices, we give a short recapitulation of the main physical characteristics of and relevant routing techniques in MANETs since these are both crucial for our choices.

11.3.1 MANET Characteristics

When using MANETs it is important to be aware of the limitations of the Wi-Fi IEEE 802.11 [7] standard which makes up the connections. The standard has a maximum bandwidth of 54 Mbit/s and is suffering from collisions with other standards and appliances like Bluetooth, cordless phones and microwave ovens. This becomes an even bigger problem when there are many nodes running in the same sending area. Since Wi-Fi shares the same restrictions as any other radio technology that sends and receives on the same frequency, it is physically required to operate in simplex. This implies a strict control of who is sending and how the radio frequency is used. The standard provides collision avoidance techniques through the use of coordination messages.

11.3.2 MANET Routing Techniques and Challenges

Routing protocols in general deliver two important services: The establishing of routes for transferring packets between nodes in the same network partition, and topology information, i.e., how nodes in a network partition are connected. Both these services are concerned with the establishing of *space paths*. OLSR [6] is an example of a space path routing protocol. OLSR uses an overlay network to speed up dissemination; nodes in the overlay gather local topology and distribute it.

In *space/time path* routing protocols a node A can pass a message to a node B belonging to a different partition if the system can retain the message for a while, and later, when A gets within reach of B, forward it to B. An interesting class of space/time path protocols is message ferrying [8, 9], where some nodes act as ferries that carry messages between different network partitions. However, since this requires the ferries to follow a preplanned movement scheme, the approach is not applicable to MANETs in general. In case of the rescue operation application domain we must provide solutions that do not rely on pre-planned movement schemes.

Another class of space/time path protocols is epidemic routing [10]. Epidemic routing provides a controlled flooding of packets and can e.g. be used for sending a message to a recipient without known address. A message is spread throughout the network until the message reaches its destination or a time-to-live counter becomes zero and the message is dropped. It relies on intermediate nodes to provide large enough message buffer space to carry all required messages. In general the overhead from wrongly directed or misguided messages consumes both buffer space and network capacity. In the case of metadata propagation, however, *all* nodes are possible addressees. The use of epidemic routing for this purpose has some definite advantages to more basic flooding algorithms: Since the message buffer is examined at node encounters, the node state is factually taken into account before retransmission. In our case aggregated metadata plays the role of the message buffer. The propagation protocol can take into account the node state when deciding how to proceed the propagation, like halting further propagation because the metadata did not change.

Other space/time path techniques include probabilistic routing and location-aided routing. Probabilistic routing techniques [11] are more relevant for the Data Synchroniser component than for GMDM's metadata propagation. In location-aided routing [12] route discovery is optimised by utilising the nodes' GPS positions. Since we cannot in general rely on the use of GPS, e.g., inside buildings and in train tunnels, we have to base our core solutions on the absence of such services.

11.4 Metadata Propagation

For the design of metadata propagation protocols we use ideas from epidemic routing. To propagate metadata throughout a network partition, each node is required to act in a router-like fashion and pass on metadata to other nodes. This can be done by repeatedly synchronising metadata contents of pairs of neighbour nodes, which has the side effect that the nodes in addition to routing metadata information, also listen in and can gather metadata for their own purpose. In this section we present three metadata propagation protocols. They all share the same node synchronisation mechanism and make use of the same kind of triggers for invoking the protocol; we therefore first present these elements before proceeding with the ideas and details of each protocol.

11.4.1 Synchronisation

A core element common to all epidemic routing algorithms, is how pairs of nodes synchronize their message buffers, or in our case, metadata contents. A basic protocol that is used in e.g. [10], works as follows: (i) Node A makes an overview (a hash) of its metadata items. A sends the hash to node B. (ii) Node B sends back the metadata items that B has, but that are not mentioned in the hash from A, plus a list of the items that B requires from A to get a complete set. (iii) Node A responds with the items required by B. This protocol works well when there are differences between the nodes. At rediscovery of an already synchronised node, some communication is needed to identify this situation.

11.4.2 Triggers

We use local triggers at a node to invoke appropriate sub-protocol instances. Triggers are needed (1) on the event of a new table replica being created locally, (2) when a new one-hop neighbour appears, and (3) when new metadata is received from another node as a part of follow-up actions on events (1) and (2) elsewhere in the network. The Data Allocator sub-component fires a trigger of type (1) when it creates a new table replica locally. As a result a new instance of the metadata propagation protocol will be invoked; the actual actions taken depend on the protocol. Triggers of type (2) allow metadata to propagate further at network merges. The CRT component is responsible for reporting the encountering of new neighbours. When the trigger is fired, the node will contact the new neighbour and synchronise its metadata contents with it. GMDM will itself fire a trigger of type (3) when new metadata is received as part of the synchronisation with another node. Here too the actual actions taken depend on the protocol.

11.4.3 Metadata Propagation Protocols

We now proceed with a description of the three protocols. Further details can be found in [13].

The Simple Protocol

Our first protocol is a fairly straight-forward implementation of epidemic routing. The protocol will try to pass a node's metadata items to neighbour nodes in a unicast fashion. The protocol is initiated each time one of the triggers are fired, i.e., at the creation of new table replicas locally, at the event of a new neighbour, and at the reception of new metadata items from a neighbour. The node in the first case approaches all its neighbours, in the second case only its new neighbours, and in the third case all neighbours except the one that provided the metadata item.

The node initiating the protocol (the 'initiator') starts by sending to each targeted neighbour (the 'listener') an overview message containing a synopsis of its metadata items. The listener responds with a message containing those of its metadata items that are not mentioned in the synopsis and that are thus not known to the initiator node, and a shortlist of the synopsis indicating which metadata items are needed to make its own metadata storage consistent with the initiator's. If the message is not received by the initiator within a timeout period, the protocol logs the result and terminates gracefully. This happens e.g. if the initiator moves out of reach before a return message is sent. The initiator concludes by sending the requested metadata items. When the requested metadata items are received or no message is received within a timeout period, the protocol terminates. If no metadata items are requested, this last step is skipped and the protocol is terminated directly.

The Broadcast Protocol

Epidemic routing uses one-to-one communication between synchronising parties. However, in radio networks every node eavesdrops on every message sent within its communication range, and then discards messages for which it is not an addressee. This can be utilised by sending one one-to-many message for getting in touch with all neighbours at once, thus reducing the number of messages eavesdropped on at non-addressee nodes. In the Broadcast Protocol the initiator node therefore initially broadcasts an overview message. Only those listeners that receive the overview message and either have additional metadata items or need some of the elements, will respond. The initiator will treat each response separately and send requested metadata items. If these do not arrive at the listener within a time frame, a timeout occurs and the protocol terminates.

If there are more than one responder to a broadcast message, the reduction on the network load is equivalent to n–1 of the Simple Protocol's overview messages both in size and transfer time, where n is the number of responders.

The Semantic Protocol

The Broadcast Protocol tends to give a propagation pattern similar to that of a stone thrown into a pond with islands (nodes). In general the outskirts of the network partition get new metadata last. Thus, if the network tends to be brittle at the edges, nodes breaking away from the partition may not get new metadata in time to carry it with them to other partitions. To break this propagation pattern we have developed the Semantic Protocol, where the idea is to do a broadcast propagation from multiple points in the network simultaneously. These points can be chosen in different ways, e.g., nodes that have few connections to other nodes in the network, or hot spot nodes. A more general approach is to construct the group of such nodes from information about semantically related nodes, e.g. the group consisting of all rescue team leaders. Theoretically this will give group members higher dissemination priority. As a side effect it gives multiple starting points for the remaining dissemination process and potentially faster overall dissemination; for instance, if the network is about to partition, the dissemination may reach some of the partitioning nodes in time to allow the dissemination to spread into other partitions.

The two phases of the Semantic Protocol propagation are (i) group synchronisation and (ii) neighbour synchronisation. In the first phase the initiator node synchronises with each of the group members by using multi-hop routing. In the second phase each of the group members independently starts up the Broadcast Protocol, and the group members consequently work in parallel on propagating changes.

11.5 Test Setup

Proof-of-concept applications that use the MIDAS prototype furnished with the broadcast protocol have been successfully exposed to field tests. In this paper, however, we present quantitative, reproducible results from tests using NEMAN [2], an environment that emulates the communication layers of the network stack in MANETs.

11.5.1 Metrics

We need suitable metrics to measure (i) correctness, i.e., that the protocol disseminates metadata to all nodes, and (ii) communication cost, i.e., the network load.

To test the correctness of the protocols we set up scenarios that each act out a well-defined dissemination phase which will be the subject of post analysis of each protocol. The number of table replicas that will be created in each scenario, is known in advance. Also known a priori is which metadata items should be stored at each node when the dissemination phase is completed. Thus, one measure is the *metadata count*, the accumulated number of metadata items stored throughout the network. To see how propagation proceeds, we can look at how the metadata count evolves over time. Since the number of propagated metadata items at each node is well-defined for stable states, we can use the metadata count to detect when a dissemination phase begins and completes.

Possible metrics for evaluating the strain that the protocols put on the network, are the *message count*—the number of protocol messages sent, and the *message size*—the accumulated number of bytes in messages sent. For measuring bandwidth, the message size is a more accurate measure than the message count. We shall see in Subsection 11.6.1 that the message count reflects fairly well the message size. Since the message count is much simpler to record than the message size, we use the message count throughout our measurements.

11.5.2 Test Scenarios

To expose the behaviour of the protocols we employ four test scenarios, two static—the Chain and Grid scenarios—where the network's connectivity remains unchanged after a start-up phase, to test basic functionality and behaviour of the protocols, and two dynamic—the Merge and Message Ferry scenarios—to test network partitioning and merge. At start-up every MIDAS node contains a table replica used for bookkeeping purposes. Since metadata about these initial replicas will spread throughout a network partition as part of the ordinary metadata propagation, after a start-up phase the metadata count will be m^2 where m is the number of nodes in the partition. Likewise, a table replica creation at a node will in time cause the number of metadata items to increase by one at every node in the partition, giving a total of $m(m+1)$ metadata items. For the static test scenarios we inject a new table replica into one node in the network to trigger such a dissemination phase. For the dynamic scenarios it is sufficient to study the dissemination phase triggered by network merge and consisting of further propagation of the initial metadata items as a result of the merge. The four test scenarios are:

[Chain] The chain protocol represents a worst case scenario wrt. connectivity and number of hops. During the start-up phase 10 nodes form a chain where all internal nodes have exactly two neighbours within direct communication range. Initial metadata information is then exchanged throughout the chain. After $35s$ the leftmost node creates a new table replica. Metadata for this replica is then propagated throughout the chain according to the protocols. In the Semantic Protocol the second and next to last nodes form a group.

[Grid] The grid scenario exposes scalability properties of the protocols. In this scenario 5×4 nodes form a grid where each inner node has exactly four neighbours within direct communication range. After $60s$ node 1 creates a new table replica.

In the Semantic Protocol two of the nodes neighbouring the main diagonal's corner nodes, form a group.

[**Merge**] The purpose of this scenario is to stress the message trigger load. It consists of two network partitions that move towards each other, get in contact at 60*s*, and continue to move until they totally overlap. At the end of the scenario all nodes have new neighbours. Each partition consists of a 3×3 grid where all nodes in a partition are within direct communication range of each other. In the Semantic Protocol the two nodes forming the group initially belong to different partitions and are not the ones that meet at the first encounter during the merge. Each of the partitions keep their original intra-node formation throughout the merge.

[**Message ferry**] This scenario introduces a network partition followed by a merge. It consists of two static partitions of 7 and 10 nodes respectively, and a ferry node moving from within the 7-node to the 10-node partition. After an initial phase the ferry starts moving, collecting metadata from the first partition as it moves. It loses connectivity with the 7-node partition at 40*s* and connects with the 10-node partition at 47*s*, stopping only after reaching the opposite side of its entry point. In the Semantic Protocol the ferry and the centre of the 10-node partition form a group.

11.6 Test Results

Table 11.1 shows for each test scenario and protocol the average number of protocol-related messages sent from a node during the dissemination phase of the scenario. Also, for comparison we include the average number of messages addressed to and received by each node, i.e., messages actually received by the CRT component and forwarded by it to the MDS component. We will comment on the results in Table 11.1 as part of the discussion in Subsection 11.6.2.

Table 11.1: Message count during the dissemination phase

Test scenario	Protocol	Avg. no. of msgs sent	Avg. no. of msgs received
Chain	Simple	29,1	25,3
	Broadcast	4,6	2,8
	Semantic	3,1	4,7
Grid	Simple	62,5	57,0
	Broadcast	3,1	6,2
	Semantic	3,7	6,8
Merge	Simple	48,1	48,1
	Broadcast	18,6	185,9
	Semantic	29,0	44,5
Ferry	Simple	21,8	21,8
	Broadcast	3,8	17,7
	Semantic	6,6	12,1

11.6.1 Comparison of Metrics

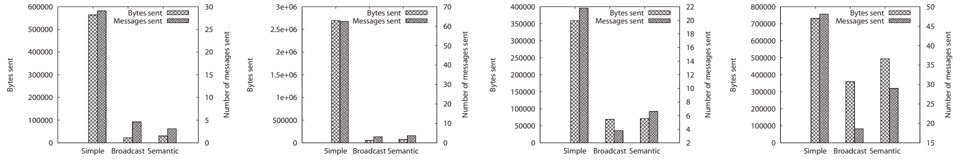
Since the message count obviously at best is an approximation to the message size, we have performed a comparison of the message count and message size to see how well the former reflects the latter.

We have obtained message sizes during the dissemination phases of each scenario by doing post analysis on TCP dumps. The start of the dissemination phase is identified by looking for control packets from the emulator GUI in NEMAN, to find for the static scenarios when the new table replica was introduced during the scenario, and for the dynamic scenarios when the first message is sent between the two first nodes that encounter each other during the network merge. We have then added up all packet lengths, excluding routing daemon traffic and control packets. Table 11.2 displays the results.

Table 11.2: Traffic measured during the dissemination phase

Test scenario	Protocol	Bytes sent
Chain	Simple	563727
	Broadcast	22115
	Semantic	30188
Grid	Simple	2693948
	Broadcast	53895
	Semantic	76641
Merge	Simple	731260
	Broadcast	360067
	Semantic	494226
Ferry	Simple	359102
	Broadcast	68983
	Semantic	71226

Figure 11.1 shows the relationship between message count and message size, with message size and message count on the left and right y-axes respectively. In general the numbers indicate that the message count is a fair measure of the number of bytes transferred. Two notable exceptions are in the chain scenario, where



(1) Chain scenario (2) Grid scenario (3) Merge scenario (4) Ferry scenario

Figure 11.1: Relationship between message count and message size

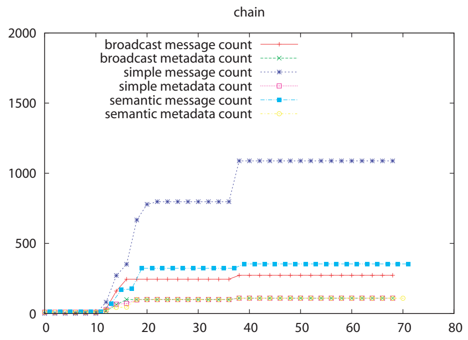


Figure 11.2: Chain scenario message and metadata count

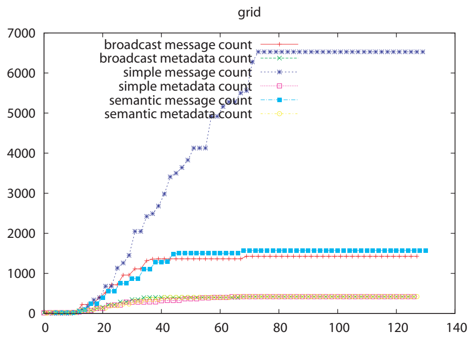


Figure 11.3: Grid scenario message and metadata count

the Broadcast Protocol sends more bytes and fewer messages than the Semantic Protocol, and in the merge scenario, where the Broadcast and Semantic Protocols have about the same byte count, but the Broadcast Protocol uses less messages than the Semantic protocol. In both cases the Broadcast protocol tends to send fewer but larger messages than the Semantic Protocol. Thus a comparison of the Broadcast and Semantic protocols based merely on message count is in general slightly misleading. Still the message count gives an indication of the active use of bandwidth, thus a reduction in the message count tends to correspond to a reduction in the message size.

11.6.2 Dissemination Results

Figures 11.2-11.5 show how the metadata counts and message counts evolve throughout the scenarios for the different protocols. The x-axes show the number of seconds from the start of the protocol. The y-axes display the counts. As the graphs show, each scenario has an initial phase after which the two counts stabilise. This is followed by the dissemination phase, after which the counts again stabilise. In the discussions below we analyse only the dissemination phase.

We have included in the graphs the metadata counts for all protocols even though they are not easily distinguishable from each other in the figures. Still they show that the metadata counts in each case reach the predicted level, demonstrating that the metadata is disseminated properly. The metadata counts reach their final level before the message counts do, i.e., the actual dissemination phases end before the message exchange completes.

In the chain scenario (Figure 11.2) the Simple Protocol during the dissemination phase sends 9 times more messages than the Semantic Protocol and more than 6 times more than the Broadcast Protocol (cf. Table 11.1). These numbers are supported by the number of bytes transferred (Table 11.2). The Broadcast Protocol sends more messages but uses less bandwidth than the Semantic Protocol. The reason might be that the Broadcast Protocol requires the use of more overview

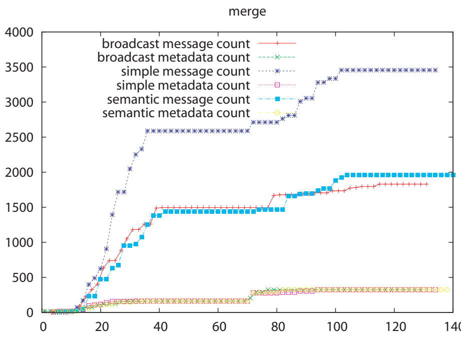


Figure 11.4: Merge scenario message and metadata count

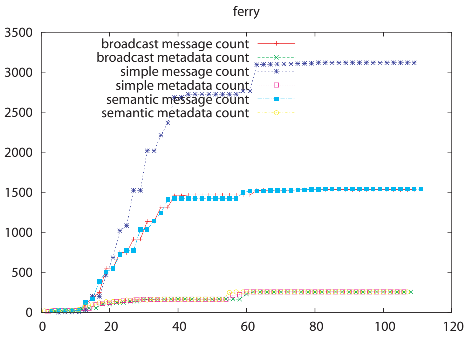


Figure 11.5: Ferry scenario message and metadata count

messages than the Semantic protocol, which means that the Semantic Protocol's multiple starting points for broadcasting in this case is more efficient than plain broadcast wrt. redundant overview messages. We expected the Semantic Protocol to use more bandwidth than the Broadcast protocol; the chain scenario confirms this.

In the grid scenario (Figure 11.3) the Simple Protocol displays its worst performance of all scenarios, both in terms of absolute numbers and relatively to the other two protocols. This is supported by the byte count in Table 11.2. The Broadcast and Semantic protocols display almost the same message counts, the Semantic Protocol being slightly higher. They show almost the same ratio as in the chain scenario wrt. the number of bytes transferred. The differences in the message counts reflect how the protocols handle high connectivity. In this scenario the average connectivity is 3.2 neighbours, thus the broadcasting of overview messages in the Broadcast and Semantic protocols shows its advantage. We had expected the Semantic Protocol to show a gain over the Broadcast Protocol in terms of message count, this was not confirmed; to reveal such a gain we may need a scenario with a larger distance between the group members. It is however difficult to obtain larger scenarios in NEMAN due to the use of Java in the project, which puts a limit on the number of nodes to 30–50.

Figure 11.4 displays the results of the merge scenario. This is the scenario with the highest neighbour ratio; during the merge phase the massive connection changes stress the triggers. It is also the scenario where the difference between the Simple Protocol and the others is lowest. Another noticeable result is the high number of messages received in the Broadcast protocol (cf. Table 11.1). This is a direct effect of the high neighbour ratio, which causes each broadcast message to contribute to a high received message count. It also causes an increase in dissemination paths, and as a consequence, an increase in redundant overview messages. We had expected the Broadcast Protocol to display relatively lower bandwidth than the others, instead the differences are less than in the other scenarios.

The ferry scenario (Figure 11.5) is actually a three-phase scenario consisting of an initial phase where each partition stabilises, a disconnection phase where the ferry leaves the smaller partition (this phase does not affect the counts), and a merge phase when the ferry enters the larger partition. In this scenario the Semantic Protocol sends almost twice as many messages as the Broadcast protocol; this is a difference not seen equally well in the other scenarios. The byte costs are however almost the same for the two protocols.

11.6.3 Group Dissemination Results

The Semantic Protocol does in general not show an improvement to the Broadcast Protocol. However, it is still of interest whether the Semantic Protocol gives a dissemination priority to the group members. For this purpose we have investigated the relative dissemination speeds of the nodes in each scenario for the Semantic Protocol. In Figure 11.6 we provide the results in case of the merge and ferry scenarios. The y-axes show the message count on a per node basis during the dissemination phase. In the merge scenario the two group members (nodes 1 and 10) originally belong to different partitions and are not the first to be involved in the merge process. They receive all new metadata from one message, which explains the direct jump from metadata count 9 to 18. Node 10 (shown with a bold line) is the last of the group to reach its new metadata count, but does so well before most of the other nodes. In the ferry scenario the ferry (represented by the line that jumps from count 8 to count 18) is part of the group; the other group member (node 2, shown with a bold line) belongs to the partition into which the ferry merges. Even though the ferry carries all new metadata, it still does no reach its new metadata count until the other group member does, and only slightly before the rest of the nodes complete. The reason is that the ferry completes all of the protocol, including the broadcasting phase, before it performs any local updates. The figure confirms that inter-group dissemination gives a priority to the group members in the merge and ferry scenarios. In the two static protocols the group members show no improvement over the other nodes (graphs are not included).

11.7 Discussion and Conclusion

We have designed, implemented and performed some initial measurements of three protocols for metadata propagation in MANETs. The protocols were designed under the assumption that all nodes store and maintain all globally shared metadata; the choice of metadata is however made carefully to keep the amount small. In general the Broadcast Protocol shows the best performance results. The Semantic Protocol is comparable to the Broadcast protocol, but in addition provides dissemination priority to a group of nodes. The Simple Protocol, a straight-forward implementation of epidemic routing, scales considerably worse than the other two. The correlation between network topology characteristics and dissemination patterns has to be analysed further to see how the dissemination pattern changes with the protocol and topology.

An issue that remains to be studied, is the impact of broadcast storms [14]. Some redundant broadcasts are avoided because each node considers its already

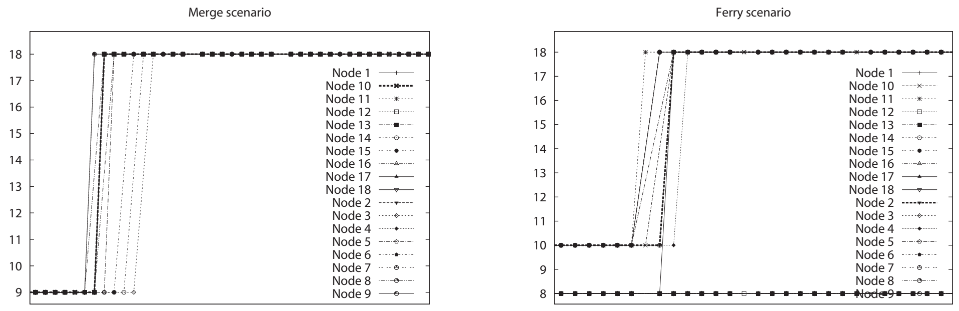


Figure 11.6: Metadata dissemination on a per node basis
(the bold lines represent the last of the group members to receive all metadata)

stored metadata before deciding whether to broadcast received metadata to further nodes. Even so the protocols perform a number of redundant messaging, thus techniques that enhance performance, like back-off timers, distance from the original broadcaster, and cluster-based techniques, should be considered. In addition, contention and collisions remain a problem. The protocols are also not fail-proof in the presence of message loss. We might introduce an additional trigger that occasionally floods metadata; on the other hand, allowing a certain amount of redundant metadata propagation might be sufficient to cover for lost messages. Thus one should balance between optimizing on redundant rebroadcasts and robustness against lost messages.

All protocols have been implemented in the middleware layer. Moving some of the functionality into OLSR might prove an advantage, for instance to utilise OLSR's overlay network to speed up dissemination.

A major factor in determining the size of the metadata storage at each node is the number of shared table replicas. The size of each metadata item is small; in average the size at each node will be in the range of only a few kilobytes and thus remains manageable even on resource-weak devices.

Acknowledgment

This work has been performed in the context of the MIDAS project, funded by the European Commission's 6th Framework Program, Contract no. 027055.

References

[1] J. Gorman, "The MIDAS project: Interworking and data sharing," in *Interworking 2006*, Santiago, Chile, January 2007.

[2] M. Pužar and T. Plagemann, "NEMAN: A network emulator for mobile ad-hoc networks," *ConTEL 2005: Proceedings of the 8th international conference on Telecommunications*, vol. 1, pp. 155–161, June 2005.

[3] T. Plagemann, E. Munthe-Kaas, K. S. Skjelsvik, M. Pužar, V. Goebel, U. Johansen, J. Gorman, and S. P. Marín, "A data sharing facility for mobile ad-hoc emergency and rescue applications," in *ICDCSW '07: Proceedings of the 27th International Conference on Distributed Computing Systems Workshops (The First International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS'07)).*1em plus 0.5em minus 0.4emWashington, DC, USA: IEEE Computer Society, 2007, pp. 307–316.

[4] T. Plagemann, E. Munthe-Kaas, and V. Goebel, "Reconsidering consistency management in shared data spaces for emergency and rescue applications," in *BTW-MDM 2007, Model Management und Metadaten-Verwaltung, workshop under GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web*, Aachen, Germany, March 2007.

[5] N. Sanderson, V. Goebel, and E. Munthe-Kaas, "Metadata management for Ad-Hoc InfoWare – a rescue and emergency use case for mobile ad-hoc scenarios," in *ODBASE05 – On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, vol. 3761.1em plus 0.5em minus 0.4emSpringer Netherlands, 11 2005, pp. 1365–1380.

[6] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," *IEEE INMIC 2001: Technology for the 21st century. Proceedings of the IEEE international Multi Topic conference*, pp. 62–68, 2001.

[7] 802.11 working group, "IEEE 802.11b 1999: Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," IEEE, 1999.

[8] M. M. B. Tariq, M. Ammar, and E. Zegura, "Message ferry route design for sparse ad hoc networks with mobile nodes," in *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing.*1em plus 0.5em minus 0.4emNew York, NY, USA: ACM, 2006, pp. 37–48.

[9] W. Zhao, M. Ammar, and E. Zegura, "A message ferrying approach for data delivery in sparse mobile ad hoc networks," in *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing.*1em plus 0.5em minus 0.4emNew York, NY, USA: ACM, 2004, pp. 187–198.

[10] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," *Duke Technical Report CS-2000-06*, July 2000.

[11] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 7, no. 3, pp. 19–20, 2003.

[12] Y.-B. Ko and N. H. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks," *Wireless Networks*, vol. 6, no. 4, pp. 307–321, Sep 2000.

[13] A. Johannessen, “Information sharing in mobile ad-hoc networks using a global metadata manager,” Master’s thesis, Department of Informatics, University of Oslo, April 30, 2008.

[14] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, “The broadcast storm problem in a mobile ad hoc network,” in *MobiCom ’99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*.1em plus 0.5em minus 0.4emNew York, NY, USA: ACM, 1999, pp. 151–162.

Chapter 12

Information Sharing in Mobile Ad-Hoc Networks: Evaluation of the MIDAS Data Space Prototype

Authors: Matija Pužar, Katrine Stemland Skjelsvik, Thomas Plagemann, Ellen Munthe-Kaas

Affiliations: Department of Informatics, University of Oslo
{matija, katrins, plageman, ellenmk}@ifi.uio.no

Publication: The 2nd International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS 2009), Toronto, Canada, June 2009

Abstract: Information sharing in dynamic mobile ad-hoc networks is a challenging task. High data availability in the presence of short and long term disconnections can be obtained by replicating shared data. The number of replicas must however be balanced against the cost of consistency management. In the MIDAS Data Space (MDS) we use optimistic replication together with internal versioning of data; this allows application-specific conflict resolution when reconciling replicas at network mergings. We have made a proof-of-concept implementation to perform experiments and to demonstrate through real-life field tests the usefulness of our design. In this paper we report our results. We have conducted a number of experiments on a small network formed by real devices to obtain a detailed performance evaluation. Using an emulation environment we have analysed and quantified the cost of consistency management, the impact of MDS operations, and the relationship between data availability and replication.

12.1 Introduction

Mobile Ad-Hoc Networks (MANETs) can provide a useful basis for data sharing in dynamic scenarios such as emergency operations or larger sports events. The main objective of the MIDAS project [2] is to define and implement a platform to simplify and speed up the task of developing and deploying highly specialised mobile services for such scenarios, where networks may consist of a variety of devices with regards to available resources. One central element of this platform is MIDAS Data Space (MDS [4]) for asynchronous information sharing. MDS is designed to work on mobile devices that together form potentially highly dynamic and unstable MANETs. MDS provides to its users a programming interface that allows the applications to use SQL operations towards the data space. MDS users can create tables, insert data, update data, delete data, and retrieve data without caring where the data is physically stored. As such it resembles a distributed relational database over MANETs. Internally, MDS addresses the various challenges that arise from the need to realise this transparency over dynamic MANETs. Especially the possibility of short term and long term disconnections (i.e., network partitions) introduces hard problems, like guaranteeing high availability of data through optimistic replication and managing the consistency of replicas. In order to enable application-specific conflict resolution, a major design decision for MDS is to perform internally versioning of data instead of updating data.

One important result of the MIDAS project is the proof-of-concept (POC) implementation of the MIDAS platform and of two demo applications that demonstrate through real-life field tests the usefulness of these results. Therefore, the main goal of the MDS POC implementation is to provide a stable working platform for higher level MIDAS components and applications. All MDS core concepts, including metadata management, optimistic replication, versioning etc., are implemented in this prototype. To keep the necessary implementation effort at an acceptable level, we have chosen to implement simplified solutions for some design alternatives; this clearly leaves some space for improvement.

On the conceptual level, systems related to MDS include information systems for disruptive environments, like Bayou [8], ROAM [7] and DHTR [9]. All of these also use optimistic replication. A detailed comparison of these related systems and MDS can be found in our earlier work describing the MDS architecture [4] and the MDS approach of versioning and consistency management [5]. The purpose of this paper is to report on the use of the MDS POC implementation for a detailed performance evaluation of a real system. This evaluation comprises experiments with real devices forming smaller networks, in order to gain real performance numbers, such as response times of local and remote MDS operations, the impact of different devices on the performance, and the overhead introduced by the versioning approach. Furthermore, we use an emulation environment to analyse the efficiency of MDS operations and to study and quantify the relationship between availability of data, number of replications, and costs of consistency management. These experiments are performed with different MANET topologies to also understand the impact of mobility, number of neighbours, and connection loss.

The rest of this paper is organised as follows: In Section 12.2, we briefly describe the main parts of the MDS design and provide some concrete examples of how versioning and synchronisation are carried out. The MDS prototype implementation is described in Section 12.3. In Section 12.4 we present our testing approach, scenarios and metrics, and present and discuss the results. Concluding remarks are given in Section 12.5.

12.2 MDS Design

From the point of view of the applications, MDS has been designed to resemble as much as possible a relational database. However, due to the nature of sparse MANETs, MDS imposes some distinctive differences compared to a traditional relational database system.

The data in the MDS data space is physically distributed across the network. Ideally, every application, regardless of the node it resides on and where the data is actually stored, should have the same view of the data. However, in the presence of disruptive and unstable networks, not all nodes might be available all the time. To obtain higher data availability, MDS can replicate existing tables, which means that replicas must be kept synchronised in order to provide the same view to all applications. Applications in different network partitions may however experience different data if each partition contains a replica and updates its contents independently. If two partitions later merge, it is not always obvious if and how data can be synchronised. To allow application-specific conflict resolution, MDS supports versioning. Versioning implies that no data is actually updated. Instead, a new version is created and the previous version is marked as updated. Likewise, no data is actually deleted, but instead only marked as deleted. Applications can choose whether or not to have a transparent view on data versioning: unless explicitly stated otherwise, only the currently valid versions are returned within a result set (see Section 12.2.2 for details on record types). When MDS receives a query from an application, it first rewrites the query to support versioning, and then sends it to the appropriate location(s), either directly to the database on the local node, or as a remote query if the table in question is stored remotely.

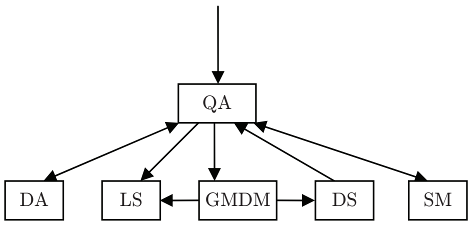


Figure 12.1: MDS Architecture

12.2.1 Architecture

The MDS architecture, including the interaction between its sub-components, is shown in Figure 12.1. The *Query Analyser* (QA) is the façade component of MDS and receives queries from the applications or from other MIDAS components (collectively referred to as MDS users). In the network, there may exist one or more replicas of a certain table. The *Data Allocator* (DA) initiates the allocation and deallocation of table replicas. Information about the whereabouts of these replicas is stored in a shared metadata table which is maintained at every node. It is the task of the *Global MetaData Manager* (GMDM) to keep this table up-to-date by propagating new metadata information and synchronising metadata tables after a network merging [3]. The *Data Synchroniser* (DS) makes sure that data in replicated tables is also synchronised. Additionally, MDS has a subscription-notification functionality taken care of by the *Subscription Manager* (SM). SM alerts MDS users via registered callback functions when some change in data matches a stored subscription.

12.2.2 Versioning

When an application wants to update a record already present in the database, say of a patient, instead of altering the data of the original record, MDS inserts a new version of the record, and the original record is marked as *Updated*. The automatically generated ID field is used to find versions of the same record and is kept unchanged when new versions are created. However, it could happen that two INSERT operations concerning the same patient are performed independently at two different nodes if the nodes are not within reach of each other at that moment. These records are then assigned different values in their ID field and are from the MDS' point of view two unrelated records. An application that wants to supersede this can define a query that allows MDS to perform a simple conflict resolution on its behalf, e.g., for a patient with a given name to choose the valid record with the latest timestamp. Most applications will only be interested in valid record versions, in such cases the other versions are preserved for auditing purposes. Applications can however choose to have invalid versions included in the result sets from queries, for instance to do conflict resolution themselves.

Versioning requires a rewriting of queries and execution of additional queries to the database. In the current implementation, rewriting is done in the following manner:

INSERT	⇨	$\text{SELECT}_M + \text{SELECT}_S + \text{INSERT}$
SELECT_n	⇨	$\text{SELECT}_M + \text{SELECT}_n$
UPDATE_n	⇨	$\text{SELECT}_M + \text{SELECT}_n + \text{UPDATE}_n + n^*(\text{INSERT} + \text{SELECT}_S)$
DELETE_n	⇨	$\text{SELECT}_M + \text{UPDATE}_n$

An index "M" denotes a query on the metadata table (to find the IDs of nodes that store replicas of the table), an index "S" is a query on the subscription table (to check for any subscriptions on data changes), while "n" denotes the actual number of processed data records (i.e., records in the result set in case of a SELECT statement, or records updated in case of an UPDATE).

As an example, an UPDATE statement where 100 records are updated, will within the MDS produce the following set of queries:

- one SELECT operation on the metadata table, to find the location(s) of the table replica
- one SELECT finding and returning the 100 records
- one UPDATE operation marking the 100 records as invalid (i.e. "Updated")
- 100 INSERT operations, one for each of the new versions to be inserted
- 100 SELECT operations on the subscription table, one for each newly inserted record

12.2.3 Synchronization

Synchronisation of replicated tables is done using two mechanisms: *eager synchronisation* and *lazy synchronisation*. An INSERT/UPDATE/DELETE request towards a replicated table is instantly (eagerly) sent to all nodes having a replica of the table after first consulting the metadata table to find the IDs of all such nodes. However, this may not be sufficient due to: a) network partitions, b) lost messages, and c) not entirely up-to-date information in the metadata table, e.g. due to a recent network merging whose effects have not yet reached this node. Therefore, lazy synchronisation is performed periodically among neighbours, and it is also triggered by the presence of new nodes in the network.

12.3 Prototype Implementation

The MIDAS middleware has been realised as a proof-of-concept implementation in two versions: POC 0.5 and POC 1.0. Implementation decisions about software usage, hardware platform and implementation environment were done based on the application domain and scenarios, but also more pragmatic considerations related to the fact that the development was carried out by different European partners.

The *software platform* used is *Linux* and *Java.* For some of the MIDAS components C++ code is also used. The *hardware platforms* we target are small, portable devices running Linux, although more powerful devices might be present as well. In our experiments, both laptops and various handheld devices were used.

A third party database is used within the Local Storage component. In order to choose a database, we made a list of requirement and picked candidates that fulfilled these for further experiments; a) small footprint, b) relational, c) support for standard SQL, and d) must be able to run on small devices. We tested the following databases: H2, Sqlite, HSQLDB and MySQL. The most interesting metrics for us are the time to load/start the database and the time to perform the different queries. The testing of the four databases was performed among others on three generations of Nokia Internet tablets, namely Nokia 770 (252 MHz), N800 (330 MHz), and N810 (400 MHz).

The experiment consisted of running respectively 1 or 100 single INSERT statements, followed by 1 SELECT/UPDATE/DELETE statement that processed the newly inserted row(s).

Figure 12.2 shows that HSQLDB and MySQL in general have the lowest response time. Having a very small footprint, and without need for compiling (which is an important factor when deploying on resource-weak devices), HSQLDB was chosen as the underlying database when evaluating the MDS prototype. Even though these measurements were not strictly measurements of MDS itself, we include them in this paper as they might be of interest to others in search for a database to be run on resource-weak devices.

12.4 Evaluation

In this section, we present an in-depth evaluation of the MDS prototype. The primary goal for this implementation is to serve as a stable working prototype for real-life demonstrations. To keep the development efforts at a feasible level, some MDS components are implemented in a simplified way and leave space for later improvements. For example, MDS sends whole Java objects to remote nodes instead of just the relevant data. To compensate for the higher bandwidth consumption in our evaluation, we compress the Java object before sending, and decompress it at the recipient. The secondary goal for the MDS prototype is to enable a detailed performance analysis of a real implementation with real devices, to identify bottlenecks and options for improvements. The insights of this evaluation are not only interesting for MDS developers, but also for middleware designers in general that focus on solutions for mobile networks.

Methodology

The dynamic nature of MANETs and the heterogeneity of devices represent a high complexity for the performance evaluation of MDS and the analysis of its results. We briefly describe the main factors of complexity and our general approach to considering them in the evaluation. The main factors we focus on here are: a) nodes, b) network, c) workload, and d) application requirements.

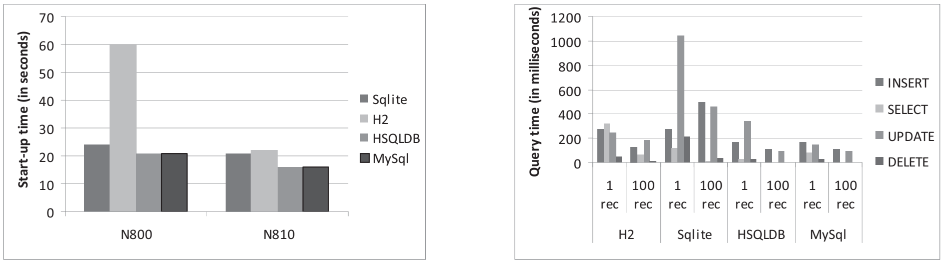


Figure 12.2: Start-up times on Nokia N800 and N810, and query times on N810

Using multiple nodes in experiments obviously increases the efforts needed for start-up, coordination, and post-gathering of results. In addition, the project assumes a variety of devices present in the network, often with different physical capabilities, resources and configurations. In our experiments, we varied both the number of nodes and the types of devices.

The assumed network is characterised through different mobility scenarios, stable and unstable connectivity, network partitions, and message loss. Therefore, we ran experiments on static networks, mobile networks without partitions, and mobile networks with partitions. To model static networks we used two topologies, first a mesh network with nodes randomly distributed in an area, and second an artificial worst case scenario for message propagation, i.e., a chain network where all nodes are lined up in a chain with a distance between the nodes such that each node has at most two neighbours (which is much less than the average number of neighbours in the mesh network).

All applications running on the nodes, including the MIDAS demo applications, produce a certain workload which in most cases varies and depends on the situation. In order to systematically study the MDS properties, in our experiments we use a synthetic workload. Two types of queries that span out the basic functionality, and thus workload, are INSERT (putting data into the database) and SELECT (retrieving data from the database), performed in different ratios. Another issue that is closely tied to the mix of applications used and their requirements, is the trade-off between performance and resource utilisation on one hand, and availability of information and reliability of the whole system on the other hand. By changing the above-mentioned parameters, we aim to get some insight into how they could be tuned to meet certain application requirements and at what cost.

There are three main options for performing experiments to evaluate the performance of the MDS prototype. These are simulation, emulation and real-life test beds. Simulation is often used because it allows running larger scale networks in non-real-time, repeating experiments, and reproducing results. The downside of simulation is that code needs to be written specifically for the chosen simulator. Emulation, in contrast to simulation, makes it possible to use the very same code that runs on real devices, in a simulated and reproducible network environment. Processes run in real time, making the server or servers' resources an important factor, and limiting the size of tested networks. Still, emulation gives a good insight into whether the written code would work as expected in a real network. Moving the code onto target devices and running in a real network is the type of experiment giving the most realistic results. However, the person efforts and hardware costs involved in performing such field tests are very high. In addition, neither the network movements nor results can be easily reproduced.

In order to benefit from the different advantages of these options, we use the two approaches of emulation and real-life test beds, and use in both of them the same MDS code. For studies of how the implementation behaves locally, as well as for smaller static networks consisting of up to 3 nodes, we use real devices. For networks of 10 nodes, both static and with introduced mobility, we use the network emulator NEMAN [6]. In this particular environment, all nodes run on a single machine, as separate processes each connected to its own virtual network interface. The processes themselves run the very same implementation as on the real devices. While being a good choice for testing functionality and behaviour in a network, it would not make sense to measure response time in such an environment as it is very dependent on the server's resources.

The experiments described in the remainder of this section are each designed to answer one of the following questions:

1. What is the general overhead introduced by MDS, and more specifically, introduced by data versioning?
2. How does MDS perform on different devices/platforms, locally and in a real network?
3. How do network and distribution affect the overall performance of the service?
4. How do different network topologies and network disruptions influence the availability of the service?

To answer these questions, the metrics we use in this performance evaluation are: 1) overhead that MDS introduces by being a layer between the applications and the local database, quantified by relating response times of MDS operations to direct operations in HSQLDB, 2) response time for operations (both local and remote), 3) resource consumption in the network, measured in number of packets and bandwidth usage, and 4) availability of the service, measured in the number of successful application queries.

One metric that has not been taken into account, is the disk usage. It is assumed that the devices' storage is not a bottleneck. The rationale for this assumption is the expected amount of data that can be generated within the limited time spans imposed by the targeted scenarios. A fact that supports this is that the size of available memory cards constantly increases while they at the same time become less expensive.

Table 12.1: Results of the MDS overhead experiments

Command	F 1	F 2
INSERT	5.4	26.8
SELECT$_1$	5.9	8.0
SELECT$_{1000}$	5.4	5.8
UPDATE$_1$	5.5	15.5
UPDATE$_{1000}$	5.0	145.5
DELETE$_1$	9.8	25.2
DELETE$_{1000}$	1.2	10.7
Average	*5.5*	*33.9*

$$F1 = \frac{MDS\_time(rewritten\_set)}{HSQLDB\_time(rewritten\_set)}$$

$$F2 = \frac{MDS\_time(rewritten\_set)}{HSQLDB\_time(original\_operation)}$$

Table 12.3: Results for local queries (in milliseconds)

Command	N810	Zypad	PC
INSERT	146	74	2
SELECT$_1$	27	35	11
SELECT$_{100}$	44	49	11
UPDATE$_1$	60	142	23
UPDATE$_{100}$	4500	3600	300
DELETE$_1$	14	67	12
DELETE$_{100}$	47	117	0

Table 12.2: Results for network queries (in milliseconds)

Dst → Src ↓	N810	Zypad	PC
N810	-	391, 173, 505, 191	166, 120, 287, 91
Zypad	358, 122, 616, 151	-	174, 91, 263, 121
PC	371, 94, 475, 277	188, 94, 319, 89	-

12.4.1 MDS overhead

To answer the first question, we performed experiments on a single Nokia N810 Internet tablet and compared time needed for a single local database query to HSQLDB directly and through MDS. Table 12.1 shows the results for 1000 records. An index “1” or “1000” denotes how many records were processed by the particular operation. The difference between the two cases is caused by the initial overhead for each query, which is negligible for queries that process a large amount of records.

12.4.2 Performance experiments on physical devices

The next set of experiments aimed at answering the second question concerning MDS’ performance on different types of physical devices, both locally and over the network. This due to the fact that in both case scenarios for the MIDAS project (a sports event and an emergency operation) it is assumed that the network will consist of a variety of devices with regards to available resources. In these experiments we used three different devices: a) Nokia N810, an Internet tablet running Linux and the Cacao JVM (400 MHz, 128 MB RAM, 2 GB Flash), b) Zypad WL1100, a wrist wearable device running Linux and J9 JVM (400MHz, 64 MB RAM, miniSD card), and c) a portable PC running Windows and J2SE JVM, (1.66 GHz, 2 GB RAM, hard disk).

First, we run local queries to see how the devices perform independently. Table 12.3 shows the average times when querying the database for *1* and *100* records respectively.

Next, we run queries over a network consisting of the same three devices, using 802.11 in infrastructure mode for easier access. Table 12.2 shows the average times for each source/destination combination. The four numbers in each cell represent the times for INSERT / SELECT$_1$ / UPDATE$_1$ / DELETE$_1$ respectively.

In general, N810 performed worst whenever network communication was involved, while for local queries it outperformed Zypad. The results show that in this particular mixed environment, the average time for a single remote query lies around 240 ms.

As it can be seen, the cost of performing remote operations is very high compared to local operations, giving an increase factor from 5 to 15, thus the applica-

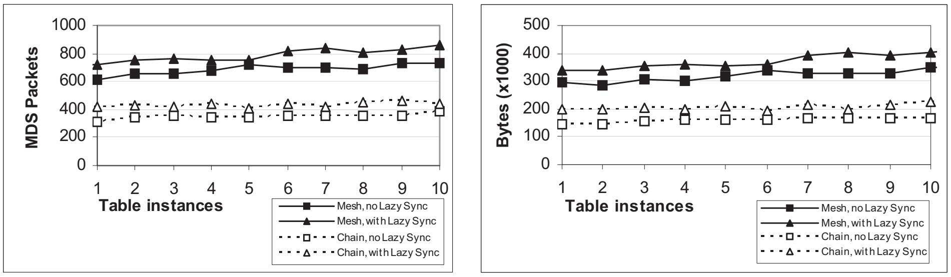


Figure 12.3: Network traffic during the initialisation process

tions' usage pattern should be carefully monitored when deciding on which node or nodes table replicas should be placed.

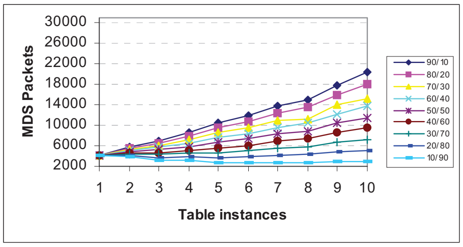
12.4.3 Emulation experiments on a static network

All emulation experiments were performed on networks consisting of 10 nodes. All queries target a single table. We varied the number of available replicas of this table from 1 to 10. In addition, there is one shared subscription table always present on all nodes. When no nodes have a replica at the beginning, the results are unpredictable as in the prototype instances are created randomly when needed, depending on which nodes started sending queries first. For that reason, in this section we omitted results with 0 initial replicas. All the experiments were run up to 10 times to verify the consistency of the results. The graphs presented here show average values.

In the following experiments we addressed the third question. We used NEMAN's monitoring channel to see how many packets are being sent between MDS instances running on different nodes, as well as how much bandwidth was used. Experiments were done both with and without lazy synchronisation being in use, to see its impact on the results.

Figure 12.3 shows how the number of replicas, and thus the amount of data needed for synchronisation purposes, affect the network usage in a mesh and a chain network (as explained earlier in the *Methodology* sub-section). The results show a constant overhead when lazy synchronisation is present. There is a slight increase in network traffic as a consequence of increasing the number of table replicas. The figure also shows that having more neighbours (mesh network) causes more traffic, as more synchronisation is present. In this, as well as in the following experiments, bandwidth usage follows proportionally the number of packets. For that reason, we omit bandwidth usage graphs in the remainder of this section.

In the next experiment, we used the synthetic workload as described earlier in the *Methodology* sub-section. Ratios between INSERT and SELECT statements were varied from 90%/10% to 10%/90%. Figure 12.4 shows separate results for the mesh and the chain network. As expected, having more direct neighbours in the mesh network increased the probability of finding data on a closer node and, con-



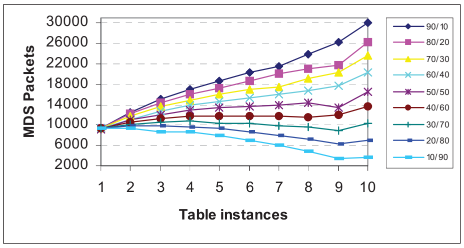


Figure 12.4: Network traffic when workload is added
(in a mesh and a chain network)

sequently the overall traffic in the network was reduced. Results also show that the usage pattern is an important factor when optimising the number of table replicas in the network. We can see that with a SELECT ratio that was high enough (approximately 10% for the mesh network and 40% for the chain network), an increased number of instances led to reduced network traffic, as the probability of finding a local replica increased. On the other hand, when the usage pattern consisted mostly of inserting data (50% or more), having more replicas only increased the amount of data being replicated in the network, without giving real benefit (except for very disruptive networks where this would pay off by increasing availability).

12.4.4 Emulation experiments on a mobile network

In order to answer the fourth and last question, we introduced mobility to see what impact changes in a node's neighbourhood have on packet loss. Within NEMAN the OLSR [1] routing protocol is used. Being a proactive routing protocol, OLSR generates periodic traffic (HELLO messages) to let the neighbouring nodes know about its presence. If a node suddenly does not hear HELLO messages from one of its neighbours, it assumes that the node might be out of reach. However, since the packet might just have been lost due to a collision, OLSR will wait for some time (i.e. 2-3 lost HELLO messages) before declaring the neighbour gone and trying to find a new, indirect route towards it.

It is in this particular period that MDS is most vulnerable and packet loss may occur. One way of reducing these periods of vulnerability might be to reduce the interval of HELLO messages. The penalty is increased overall network traffic, and thus it should be considered carefully. In a specially targeted experiment, increasing the frequency of HELLO messages from 2 seconds to 1 second gave a minimal gain, only reducing the amount of lost packets by 3 (out of a total of 600) at its best. Here we focus on packet loss experienced by applications, i.e. failed remote queries, while ignoring packet loss at the synchronisation components (GMDM and DS), as this is taken care of by the MDS and is transparent to the applications.

The usage pattern (number of INSERT vs. SELECT operations) proved not to be a considerable factor with regards to the packet loss, and results were consis-

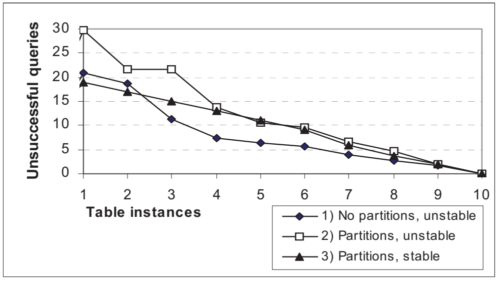


Figure 12.5: Unsuccessful queries due to packet loss

tently very similar for these. As expected, one factor that did play a big role in packet loss, was the number of table replicas in the network. If only one node has a replica, the probability of reaching it in a disruptive network will be much lower than if more nodes have it. On the other hand, if all nodes have a replica, all the queries will be performed locally and the application will not experience packet loss at all, something results shown in Figure 12.5 confirm.

We ran the experiment on a mobile network having an area of 400x300 m, where each of the 10 nodes had a 150 m wireless range. The random waypoint model was used for the mobility pattern. The same experiments were performed in three different dynamic network topologies.

In the first topology, there were frequent changes in each node's neighbourhood (a total of 350 route changes in the network were reported during the 60 s the scenario was running), but there were no partitions, i.e. all the nodes should have been reachable all the time. In the second topology, there was a network split for 15 s during which two partitions, each having 5 nodes, could not communicate. It can be seen that with a low number of table replicas, and especially when there was only one replica, the number of lost packets came to its maximum. By adding just one replica into the second partition, the packet loss was notably reduced. In the third topology, there were no changes in the nodes' neighbourhood except for a network split similar to the one in the second topology. As a control experiment we ran the same scenario with no partitions and no route changes (getting a completely static network) and, as expected, no packets were lost.

These results show that a dynamic environment does indeed affect the successfulness of the queries. However, the overall number of unsuccessful queries is relatively small compared to the total number of queries (3-5% at most). By having knowledge of the application requirements, MDS can adjust the number of replicas to achieve a good trade-off between cost and performance.

12.5 Conclusion

In this paper, we have presented an evaluation of the MDS prototype implementation. MDS replicates data in order to achieve high availability, in particular in the presence of partitioning. Inconsistency of data is handled by not deleting or updating data but instead storing a new version, and through this providing the applications the means for imposing their own conflict resolution policies.

Evaluating a distributed system prototype for use in a disruptive environment is a challenging task. We have therefore conducted both field tests and emulation experiments. In more detail, we have used a step-by-step approach, first evaluating MDS on different single stand-alone devices, then investigating response times for remote queries through varying the kind of device sending and receiving the query, and finally using network emulations and varying the topology and the application workload. In short our findings are as follows: (1) The general overhead, i.e., the response time introduced by MDS processing application queries and executing of the resulting set of internal queries, gives an average factor of 34. (2) As expected, the kinds of devices/platforms used, impact both the local and remote query response time, ranging from tens to hundreds of milliseconds. The increase of response time of remote queries compared to local queries is between 5 and 15 times. (3) By varying the ratio of INSERT and SELECT operations and the number of replicas in a static network, we observed an interesting trend revealing what impact the number of replicas has on bandwidth usage and response time. These results can be used for tuning the number of replicas based on either predicted application behaviour or by monitoring the MDS usage at run-time. (4) The availability of the data was measured by counting the number of unsuccessful queries during the experiments. In our test scenarios these vary from 0 to 5%.

The results presented in this paper show that the choice of data placement is a very important factor when considering the resource usage and the applications' perceived quality of the overall system. The results provide a good basis for our further work, where we focus on replication strategies that adapt to dynamic changes in network topologies.

Acknowledgment

This work has been performed in the context of the MIDAS project (funded by the European Commission's 6th Framework Program, contract no. 027055), and the SIRIUS project (funded by the Norwegian Research Council, project no. 186954).

References

[1] Clausen, T., Jacquet P., “Optimized Link State Routing Protocol (OLSR)”, RFC 3626, October 2003

[2] Gorman, J., “The MIDAS project: Interworking and data sharing”, Interworking 2006, Santiago, Chile, January 2007

[3] Johannesen, A., “Information sharing in mobile ad-hoc networks using a global metadata manager”, Master’s thesis, Department of Informatics, University of Oslo, April 2008

[4] Plagemann et al., “A data sharing facility for mobile ad-hoc emergency and rescue applications”, Proceedings of the First International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS 2007), Toronto, Canada, June 2007

[5] Plagemann, T., Munthe-Kaas, E., Goebel, V., “Reconsidering consistency management in shared data spaces for emergency and rescue applications”, Workshop “Tailor-made Data Management”, BTW2007, Aachen, March 2007

[6] Pužar, M., Plagemann, T., “NEMAN: A network emulator for mobile ad-hoc networks”, Proceedings of the 8th International Conference on Telecommunications (ConTEL 2005), Zagreb, Croatia, June 2005

[7] Rathner, D., Reiher, D., Popek, G.J., “Roam: A scalable replication system for mobility”, Mobile Network Applications, Vol. 9, No. 5, Oct 2004

[8] Terry et al., “Managing update conflicts in Bayou, a weakly connected replicated storage system”, Proceedings of the fifteenth ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, United States, December 1995

[9] Yu, H., Martin, P., Hassanein, H.S., “Cluster-based replication for large-scale mobile ad-hoc networks”, International Conference on Wireless Networks, Communications and Mobile Computing, Vol. 1, June 2005

Chapter 13

Evaluation of Replica Placement Strategies for a Shared Data Space in Mobile Ad-Hoc Networks

Authors: Matija Pužar, Thomas Plagemann

Affiliations: Department of Informatics, University of Oslo
{matija, plageman}@ifi.uio.no

Publication: The 13th International Conference on Network-Based Information Systems (NBiS-2010), Takayama, Gifu, Japan, 2010

Abstract: The dynamic nature of mobile ad-hoc networks (MANETs) can easily lead to data being inaccessible due to constant route changes and network partitions. One method often used for increasing reliability and availability of data is replication. However, replication comes with costs, those of transferring and storing data and keeping track of consistency between replicas. For that reason, we have identified the core factors impacting the resulting network traffic. We have performed experimental studies with a real world prototype of a distributed data management system for MANETs. Furthermore, we have done an extensive simulation study showing where table replicas should be placed in the network, in order to minimize network traffic generated by access to the databases and by the synchronization data. The results of the experiments are consistent and show that by using clustering techniques we can achieve close-to-optimal traffic by placing replicas on approximately 10 % of nodes.

13.1 Introduction

Replication is a well known method for increasing reliability and availability of data. This is especially important in mobile ad-hoc networks (MANETs), where the network topology is constantly changing. Moreover, whole groups or nodes might detach from the rest of the network, forming their own partitions. By replicating data, we minimize the probability that data is unavailable in such cases. However, this comes with costs, those of transferring and storing data multiple times, and managing consistency between replicas.

The MIDAS middleware [2] aims to provide applications an environment for sharing data, with special focus on MANETs and their disruptive nature. As part of the project, a working prototype of MIDAS has been implemented and used in two real world field tests. The MIDAS Data Space (MDS) [12] is a component in MIDAS providing a distributed data space, where the actual location of data is transparent to the applications. To the applications, MDS looks like a single ordinary relational database, with support for standard SQL queries. In reality, instances of tables (i.e. replicas) are distributed among different nodes in the network, and MDS keeps track of their location. MDS analyzes each query and, when necessary, issues remote queries to nodes in the network having a replica of the tables being queried.

One important aspect of such a distributed system is replica placement, i.e. to decide how many replicas are needed and where they should be placed, something MDS is in charge of. Well chosen placement of replicas makes it possible to minimize network traffic and provides robustness and redundancy in case of network partitioning. It is therefore important to understand how the MDS access pattern, the replica placement, and the network topology impact the network traffic. Bad placement of replicas can lead to two problems. First, too few replicas might cause read operations being forwarded to distant nodes, or important data not being accessible due to network partitioning. Second, too many replicas can lead to excess synchronization traffic which in turn can cause link saturation and, as a consequence, a part of the network not being usable at all. The same situation can happen if the locations for replicas are not well chosen. This is especially important for write operations, since changes need to be propagated to all replicas. The network topology determines the path used for forwarding packets and directly affects bandwidth usage and power usage on nodes on the path. However, due to the broadcast nature of wireless networks one must not only take the links of the end-to-end path into account, because all nodes in transmission range of all forwarding nodes are also affected. In the measurements performed in this paper, we take into account the impact MDS traffic has on nodes in close vicinity to the packets' end-to-end paths, something that is often ignored, yet important. Such collateral traffic not only uses nodes' bandwidth, but also other resources such as CPU and, as a consequence, battery power.

In order to perform good replica placement in MDS, i.e. a placement that increases data availability and decreases network traffic and energy consumption, we have developed a Traffic Controller for MDS. It monitors the network topology and the applications' access pattern to MDS in real-time, and makes decisions on replica placement based on this information. The current network topology is acquired non-intrusively from the underlying proactive routing protocol. MDS access statistics are reported constantly by all nodes hosting replicas. By analyzing access statistics and network topology, the Traffic Controller can calculate the costs or savings of potential changes in the replica placement, including the cost of transferring data to replicas. However, these relationships are quite complex and the problem space defined by the range of possible access patterns and the range of possible network topologies is extremely large. Therefore, we have performed a systematic, but still extensive empirical study varying the abovementioned factors, to determine proper replica placement strategies.

The results of the experiments are consistent and show that by using clustering techniques we can, as a rule, achieve better results than by using random placement. In the cases where workload is not known upfront, we can achieve close-to-optimal traffic by placing replicas on approximately 10 % of nodes, if chosen correctly (e.g., by using clustering techniques). We have also analyzed the costs of reconfiguration of clusters, in different cases of adaptation to changes in network topology. The results show clearly that the system should not react blindly to topology changes, as it might cause undesired network traffic. Even though this study has been made with the MDS implementation, its results are also useful for other replication problems in MANETs. To the best of our knowledge, no similar study of this complexity or with corresponding insights has been performed yet. Furthermore, we consider in our evaluation the intrinsic property of the broadcast nature of wireless networks, which is ignored in most existing works. Based on the evaluation results, we have developed a new approach for further reduction of synchronization traffic, based on a combination of warm standby and caching techniques.

The rest of this paper is organized as follows: Section II gives an overview of some related work. In Section III, we present shortly the current state of MDS, and describe in detail the Traffic Controller mechanism. The methodology used for performing the experiments is described in Section IV, followed by a description of experiments on static networks in Section V, and mobile networks in Section VI. Finally, in Section VII we give a conclusion and present ideas for future work.

13.2 Related Work

Existing work on data replication in MANETs typically takes the nodes' physical locations into consideration to determine what group or cluster a node is member of and to predict partitioning. In [11], the authors present a comprehensive survey of different replication techniques, placing them into five main categories depending on whether the techniques are power-aware, real-time-aware or partition-aware. Several techniques ([3], [16], to name a few) take into consideration nodes' physical locations to determine what group or cluster a node is member of, as well as for predicting partitioning. Even though physical location can give an indication on nodes' present and future connectivity, we argue that one must not rely solely on this information. Nodes that are physically in close proximity might for various reasons (obstacles, traffic congestion, etc.) not have network connectivity at all. In addition, target devices might not be equipped with a GPS receiver, and target scenarios might be places in locations where GPS signal might not be present (e.g. in a large building, tunnel, etc.). Supported by results in [5], we rely in our work instead on the actual connectivity for replica placement, as reported by the routing protocol. By using clustering techniques based on nodes' neighborhoods, MDS can partially predict partitions and act accordingly. MDS does not support real-time transactions, nor does it (in its current implementation) take into consideration the nodes' resources. As such, MDS can be placed into the *non-power-aware, non-real-time-aware* and *partition-aware* category. REDMAN [1] is a middleware for managing replicas in dense MANETs, defined as networks where each node has at least a certain number of direct neighbors, and where the node density in the deployment area is relatively constant during time. Like in MDS, REDMAN distributes metadata information about replica placements, however unlike in MDS, it is not distributed on all nodes. In [19], the ARAM algorithm and two enhancements are presented. The number of hops is used as the main metric of the communication cost of data access. In ARAM, nodes containing replicas decide locally on replica placements, based on periodic analysis of information on access from their neighbors. Like in MDS, both read and write operations are performed on the closest node, while write operations are then forwarded to all other replicas. Also, like in MDS, nodes hosting replicas keep access statistics locally, but unlike MDS, the decisions regarding replica placements are done locally in the ARAM family of algorithms. In [18], the same authors present a clustering-based algorithm, where nodes' position and the distance between them are used as the basis for calculations. The algorithm presented in [17] has a goal similar to ours, i.e., to minimize replication cost, given as a compound of several factors, but is developed for static networks.

Another characteristic of MDS distinguishing it from some other systems (such as [16]), is the fact that MDS does not make a distinction between a main (often called *master*) copy and secondary ones, i.e., all replicas are treated equally. In that respect, MDS can be considered as being a peer-to-peer system.

13.3 MDS and the Traffic Controller

As part of the MIDAS project, we developed a working proof-of-concept implementation of the MDS. The implementation includes a fully functioning metadata managing system [9], data versioning system, subscription and notification mechanism, as well as eager and lazy synchronization [14]. Due to time constraints, however, only very simplified versions of some of the other sub-components are present in the implementation. One of them is the Data Allocator (DA) sub-component, responsible for placing replicas in the network. In this paper, we present a solution for the DA sub-component based on a Traffic Controller mechanism.

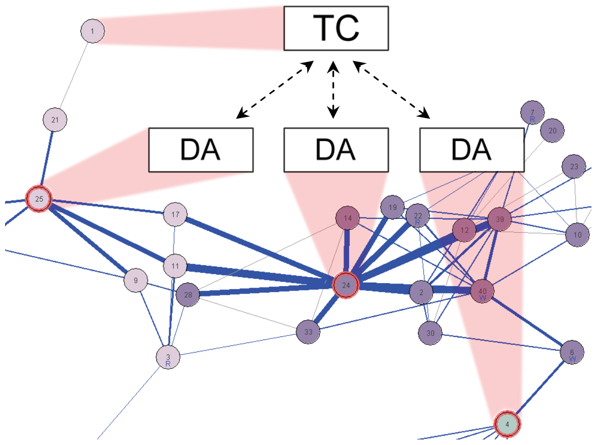


Figure 13.1: Screenshot from the test application
on top of the real MDS prototype, showing in addition the report channels towards the TC

The Traffic Controller (TC) is the node monitoring the network traffic caused by MDS usage, and the network topology, to decide where to place replicas; and by this how to control the MDS traffic in the network. The core challenge we address in this paper is how to determine good placement strategies for the possible MDS access patterns and network topologies. It is also important to make sure that there is exactly one TC instance in every partition and that all nodes in the partition know about it. A detailed description of how to assure this in dynamic MANETs is out of the scope of this paper, but we briefly sketch the basic idea of our approach. First of all, on all MDS nodes there is a TC instance, but only one is supposed to be active in each partition. Since the routing table gives any node in each partition a quite good view of the network topology, it is possible to simply choose the node with the highest ID to be the TC. The TC announces its presence by sending periodic beacons. Incomplete or outdated topology information can lead to more than one active TC instance or no instance at all, but both cases are identified by detecting irregularities in beacon announcements and can be afterwards corrected. Please note that the proper functioning of MDS is not depending on the TC; fixing a missing TC or too many TCs in one partition only shortly delays the possibility to adapt the replica placement.

Each node having a replica of one or more tables sends periodical messages to the TC with its own usage statistics. These data are grouped by tables and source nodes, and include the total incoming and outgoing traffic generated by their queries. By putting the usage statistics together with the knowledge about the current network topology, the TC can easily calculate the usage of each and every link in the network and, as a consequence, choose to do reconfiguration. This would typically be done in the case a different configuration would cause less bandwidth usage. Using the statistical data, the TC node can predict what the future bandwidth usage will be if it indeed decides to perform reconfiguration of replicas. Another case where reconfiguration might be triggered is when TC notices that a link (or more of them) became saturated. TC uses an adjustable parameter to define the threshold of when a link becomes saturated. Through this parameter, the TC can also implicitly consider non-MDS traffic in the decision making. The cost of gathering statistics at the TC node is negligible compared to the regular traffic (typically a few hundreds of bytes per packet, sent every 10 seconds), and for that reason it was not taken into consideration in the following experiments.

In order to visualize the protocols and to see how the system reacts on concrete actions, we have developed a test application that can be run on the TC node. Figure 13.1 shows a screenshot of the test application running in real time on one of the nodes in the emulated environment. The thickness of the lines represents bandwidth usage on a particular link, while a thicker round border means that a node has a replica of the table being monitored. The application illustrates the network, as well as link usage in real time, as reported by the nodes gathering statistics. It also gives the possibility to manually create or remove replicas, start and stop queries at given nodes, change clustering parameters, etc. That way, we can instantly see how the network reacts to any action. The figure shows also how the DA sub-components on the nodes having replicas communicate with the TC node.

There are two main types of queries towards MDS that cause very different effects with respect to network traffic, i.e. *read* and *write* (issued by the applications as SELECT and INSERT SQL statements, respectively). A *read* operation comprises typically a short query message and a possibly large response from the single node having the nearest replica. A *write* operation comprises a short query message (possibly large if images are inserted), a short acknowledgement from the nearest node, and the messages needed to perform instant (eager) update of all the remaining replicas within the network partition. It is important therefore to match the number and location of replicas as closely as possible to the applications' access pattern on the various nodes.

Network topology is the second factor that has a major impact on the overall network traffic. During a short period of time, the topology can be considered being static and MDS can configure the replica placement accordingly. However, mobility can cause constant reconfiguration of routes within the network, as well as network partitioning. Both of these can have a huge impact on replica placement choices. While in practice nothing much can be done after partitioning has already happened, there are several ways MDS can react to an internal reconfiguration of the network. It can decide to remove some of the existing replicas or add one or more new ones. However, the latter comes with a cost, that of copying data to a new location, so any such decision has to be planned carefully to see whether such a reconfiguration would pay off in a short or long term. The two most important parameters TC needs to consider when deciding whether to perform reconfiguration or not, can be put in the following simple equation:

$$\mathit{sync\_traffic} \leq \triangle \mathit{app\_traffic}$$

where *sync_traffic* denotes the cost of creating a new replica and synchronizing it (i.e. copying all necessary old data to it), and *△app_traffic* denotes the predicted difference in the traffic between the current status and the new placement, in a certain period of time. In short, the cost of creating a replica should not be bigger than the gain achieved by doing it. However, there are a few challenges that arise, e.g. to determine what is the period to be looked in, and then how exactly the replica location should be reconfigured.

The TC can recalculate the assumed future network traffic if any of the replicas is removed or a new is added. One way to minimize the effect of re-clustering might be to always only move the replica or replicas that have the worst statistics, i.e. cause a lot of traffic that might have been avoided if a change was made.

To understand more in-depth the relation between the network traffic and replica placement, a more thorough study had to be made. The following sections present the experiments we performed, and some conclusions we drew out of their results. These results will help us to better design the algorithms within the TC and MDS in general.

13.4 Methodology

To be able to perform a thorough study and get meaningful and reproducible results, we had to run simulation or emulation experiments. In our emulation tool NEMAN [13], we can perform network experiments on a single machine. Even though a working implementation of MDS in Java already exists, which runs in NEMAN, we could not use it for large scale experiments due to high resource usage present when running a larger number of Java processes on the same machine. Therefore, we created a simulation model of MDS and verified its correctness by comparing it in simple scenarios with the real implementation. In the simulated experiments, the most important features of MDS, i.e. remote queries and eager synchronization, are modeled to match the existing implementation. In the simulated MDS, it was necessary to determine which path (i.e. which links) a packet would go through from the source to the destination node. Since these runs were not performed in real-time, the tested scenarios were first put into NEMAN, with the OLSR routing protocol [1] running on top, to gather routing information for the simulated MDS. As a consequence, the simulated MDS was able to calculate the exact same paths the real MDS would have used in the same situation. To calculate each link's bandwidth usage, the path for each packet was followed hop-by-hop to see which links would be used when it was resent. This includes also links towards nodes which are not on the end-to-end path, but which do hear the packet due to the broadcast nature of the wireless medium.

In our experiments, we varied the following parameters: a) number and placement of readers and writers, b) number of nodes, c) size of the area, d) mobility, and e) number and placement of replicas. Links were considered to be ideal, i.e. we did not take into consideration the possible bandwidth saturation. Since it would be computationally unfeasible to study all possible combinations, we have used a systematic approach in choosing test cases. We have identified some extreme cases (e.g. placing replicas on one node on the border of the network, as opposed to placing them on all nodes), as well as some sample cases in between. We have then identified three typical static scenarios, and a mobile one. To be able to analyze the mobile scenario, we took static snapshots of the network topology every 60 seconds. The location of replicas was chosen in two ways. First, they were placed manually (for small amounts) and randomly (for larger amounts) varying from having only 1 replica at different places in the network, to having replicas on all nodes. Later, different clustering algorithms were used, where replicas were placed on the chosen cluster heads, to see the effect that clustering has on performance.

The first set of experiments was run on 3 different static scenarios, by varying the placement of replicas and application generated traffic. The scenarios are similar in that the average number of neighbors (i.e. network density) is the same, while the number of nodes and the network configuration are varied.

The first scenario to be tested (S1, as seen in Figure 13.1) was a snapshot from a scenario generated by RoboCup Rescue [8] and consisted of 40 nodes (of which 2 were disconnected at that very point in time when the snapshot was taken). In this scenario, each node has an average of 3.75 neighbors. We call this parameter network density. Since network density is an important factor for the performance of MDS, this number was used as a reference when creating the next two scenarios, to be able to compare the results. The second and third scenario (S2 and S3) consist of 100 nodes within an area of 4000x1000 and 2000x2000 respectively. The scenarios are generated by using the network simulator ns2's tool *setdest*, with minor manual adjustments to match the desired average density.

In all scenarios, the traffic was varied from having almost no traffic at all, to having all nodes performing both read and write operations. Table 13.1 shows the queries that were used in the experiments, with the respective request and response sizes. The request and response sizes were obtained from empirical experiments with real MDS and are very dependent on the queries themselves and the current status of the database, especially the request size for write queries and response size for read queries.

Table 13.1: Request and response size of the queries used in the experiments

Type	Actual query	Req	Resp
read	SELECT * FROM table LIMIT 10	90 bytes	1900 bytes
write	INSERT INTO table (descr) VALUES ('val')	220 bytes	680 bytes

13.5 Static Network Experiments

All the experiments presented in Sections 13.5 and 13.6 are based on the current MDS implementation, as described in Section 13.3. This includes the present solutions for eager and lazy synchronization. The results of the experiments are presented as graphs showing bandwidth usage (in bytes per second) for each of the direct bidirectional links in the simulated wireless network, both individually and cumulative for several experiments. In the graphical interpretation, the links are always sorted ascending by bandwidth usage to make it easier to analyze the re-

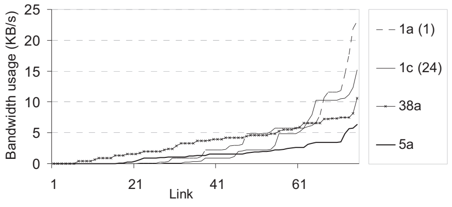


Figure 13.2: Link usage, S1, manual replica placement

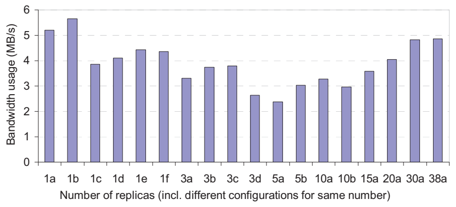


Figure 13.3: Cumulative bandwidth usage, S1, manual replica placement

sults visually. The rightmost links are the ones being used most in a single run. The surface under each line represents the total bandwidth usage in the network.

Scenario S1. This experiment consisted of runs having 18 different replica placements, each analyzed with 6 different configurations of readers and writers. Figure 13.2 shows a few representative runs (with 1, 5 and 38 replicas) for an example with 4 readers and 4 writers. The number in the brackets for the cases *1a* and *1c* denotes the node hosting the replica (node 1 is a border node, whereas node 24 is placed centrally).

In the first 6 runs there was only one replica in the network, placed on the respective node. The graph shows clearly the difference between placing a replica on a border node as opposed to placing it centrally.

The next four runs had 3 replicas in the network, followed by two runs with 5 replicas, then two runs with 10 replicas, and continuing to up to 38 replicas (i.e. all nodes having a replica). It can be seen that increasing the number of replicas reduces the individual and total bandwidth usage, but only up to a certain point. In this case, 3 or 5 well placed replicas (in Figure 13.2 we show only the case with 5, denoted as *5a*) gave the best results. Having more than 5 replicas started increasing the bandwidth usage due to synchronization traffic, with only 4 writers already (see the extreme case with 38 replicas, denoted as 38a). With more writers, this increase is even more visible.

After comparing all the six configurations of readers and writers (i.e. readers/writers: 12/1, 12/4, 12/12, 4/4, 1/12, and 38/38), the overall conclusion is that

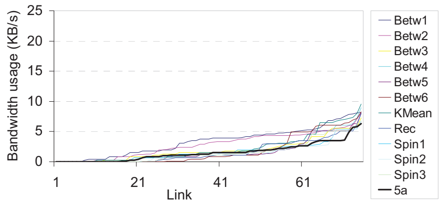


Figure 13.4: Link usage, S1, using clustering algorithms

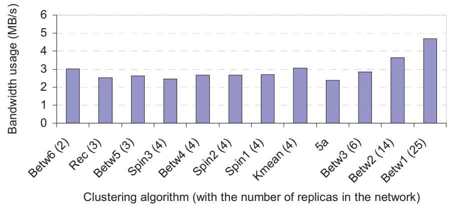


Figure 13.5: Cumulative bandwidth usage, S1, clustering algorithms

the configuration *5a* gave the lowest total bandwidth usage. The total bandwidth usage presented in Figure 13.3 is a sum of bandwidth usages in each of the 6 configurations.

In the next experiment, different clustering algorithms were tested to see the potential benefits of using clustering techniques, compared to manual or random placement. In Figures 13.4 and 13.5, we can see that the results for most of the clustering algorithms are comparable to the best results achieved with careful manual placement. To illustrate this, the previously described *5a* configuration is added to the graphs below for comparison. In Figure 13.5, the numbers in brackets represent the number of replicas in the given configuration.

We used the following clustering techniques: Recursive clustering [10], K-Mean [7], Spinglass [15] (with 3 differently chosen cluster heads) and Betweenness [2] (with maximal diameter varying from 1 to 6). The general conclusion after the experiments with scenario S1 is that, without prior knowledge of applications' access pattern (i.e. the amount of nodes reading and writing), it is safe to choose between 4 and 6 replicas in the network. In this particular scenario, this number represents 10-15 % of nodes.

Scenarios S2 and S3. This experiment consisted of runs having 10 different replica placements, each analyzed in 8 different configurations of readers and writers. Figures 13.6 and 13.7 show two very different examples, one with 1 reader and 100 writers, compared to one with 100 readers and 1 writer. The most relevant runs (i.e. those having 1, 10 and 100 replicas) are marked using thicker lines.

The line showing 100 replicas (i.e. one replica on each node) shows clearly what happened in both situations. When all nodes performed write operations, this would create enormous amounts of traffic due to data synchronization. On the other hand, when there were almost no nodes performing write operations, there was no traffic at all, i.e. any node wanting to read something had a local replica and did not need to query a remote node. What can be seen from these graphs is that having between 10 and 15 replicas gave most stable results with regard to changes in the applications' access pattern. The total bandwidth usage was lowest between 3 and 20 replicas (as seen in Figure 13.8).

Next, we used the Betweenness clustering algorithm to see the potential benefits of using clusters, compared to manual or random placement. We varied the

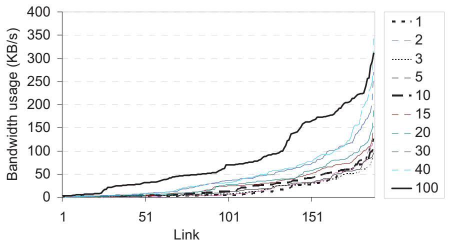


Figure 13.6: Link usage, S1, manual replica placement

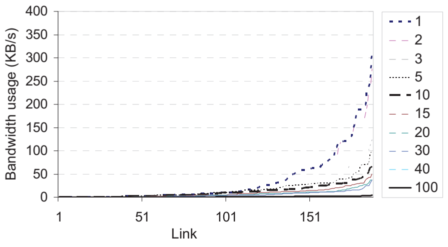


Figure 13.7: Cumulative bandwidth usage, S1, manual replica placement

maximal diameter (from 1 to 10), getting as a result 3 to 46 clusters, i.e. replicas. Figure 13.9 presents a case with 1 reader and 100 writers. With 10 replicas (denoted as Betw05 in the graph), the network traffic proved to be least susceptible to the variation in applications' access pattern.

The experiments made on scenario S3 gave results comparable to those from S2. There were only minor differences in the results caused by the different configuration of the network, which do not impact the conclusions.

13.6 Mobile Network Experiments

Mobility might cause clusters to constantly change, something that would also change the desired location of replicas. It can be a very costly process to constantly move replicas around in the network. In the following experiments we have investigated how TC can react to topology changes caused by mobility, and what the costs or gains of such actions might be.

13.6.1 Frequent Reconfiguration vs. Static Configuration

To see the effect that mobility has on data distribution and replica placement, we used the original RoboCup Rescue scenario from which the snapshot S1 was taken. This scenario (S4) is 18000 seconds (5 hours) long, of which, due to table size restrictions in Microsoft Excel, only the first 15000 seconds were considered. Snapshots were taken every 60 seconds to see the topology changes and cluster configurations at those moments, and how these affect traffic caused by the usage of MDS.

Figure 13.10 shows a comparison of the total traffic in the four cases where we had (a) the *ideal* placement of replicas achieved by calculating all possible combinations, (b) re-clustering every 60 seconds and creating new replicas accordingly, (c) random placement of replicas, and (d) the worst possible placement achieved by calculating all possible combinations. For visual clarity, only the first 5000 seconds are shown here, but the rest of the scenario follows the same trend. In all four cases, all 40 nodes were constantly performing both read and write operations. For

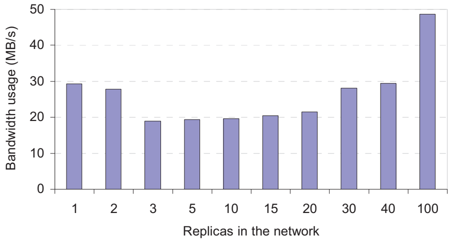


Figure 13.8: Cumulative bandwidth usage, S2, manual placement

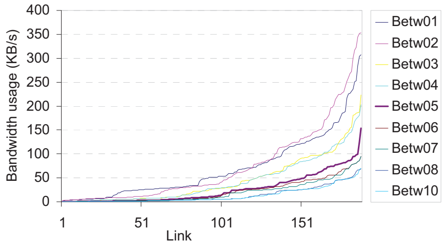


Figure 13.9: Link usage, S2, using clustering, 1 reader / 100 writers

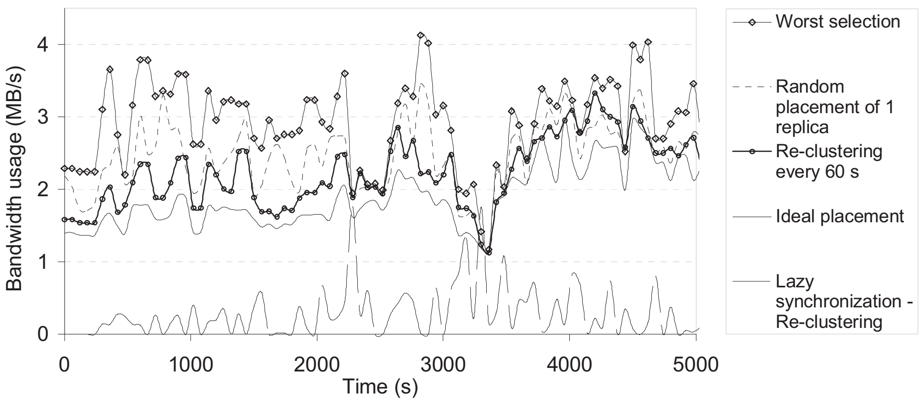


Figure 13.10: Total bandwidth usage in 4 different cases, S4

fair comparison, there was always one replica in each partition, since the clustering algorithm (Betweenness, with the maximal cluster diameter of 4) in all cases chose only one cluster (and, as a consequence, only one replica) per partition. The cost of creating new replicas and copying data into them is not yet taken into consideration.

The average bandwidth usage was about 2.5 MB/s. The average savings of having the ideal choice compared to the worst possible choice were approximately 790 KB/s. Using random placement saved approximately 480 KB/s, while performing constant clustering saved approximately 570 KB/s compared to the worst case. To put it the other way around, the random placement was 380 KB/s, while clustering was only 216 KB/s more expensive than the ideal placement. Since the ideal placement is only theoretical and might not be computationally feasible in real time, the results show that by doing active re-clustering, to adapt to the new network topology, we do achieve considerable savings in bandwidth usage if only user generated traffic is considered. However, creating new replicas comes with a cost, that of transferring old data to the newly generated replicas in order to have a consistent view on the shared space. This cost is represented by the series in the graph marked *Lazy synchronization* and is explained later.

13.6.2 Gains and Costs of Frequent Reconfiguration

The next experiment was performed to see what is the actual saving (if any) at each point in time (*t*) when clusters are re-configured, as opposed to keeping the current configuration (*t-60*). Figure 13.11 shows the results of the experiment.

At each point in time where reconfiguration was to be done, there might have been replicas that should have been removed, and there might have been new ones to be created. The ones to be removed are not taken into consideration at this time as that action does not cause any considerable traffic. The amount of newly created replicas varied from 0 to 6, with an average of about 1.91. The reconfiguration that included creation of new replicas was desirable in about 30 % of checks, i.e. there was at least 1 new replica to be created. The savings at each step varied from about 550 KB/s in one end to negative numbers (i.e. reconfiguration would be more expensive than keeping the current state) in the other. If only the positive numbers are taken into consideration, which is the only case where we benefit from making changes, we have to see what is the maximal cost of creating each new replica (i.e. maximal table size). If the table at that moment is larger than the given value, the cost of reconfiguring is larger than the short-term gain that would be achieved by doing it. In this very experiment, the average savings were about 140 KB/s. It has to be noted that, if a node already has had a replica of the table in the past, it will not be necessary to re-synchronize the whole table but only the newly added or changed records since it had been deactivated. This is possible as long as replicas are only deactivated and not physically removed, i.e., by having them in warm standby. In addition, nodes on the end-to-end path (who anyway receive all the packets) might transparently cache queries and synchronization traffic going through them. This, in combination with replicas in warm standby, might significantly reduce network traffic when a new replica is created.

Another important thing to remember is that, when a new replica is created due to partitioning of the network, this will not cause any traffic at the moment of creation since the partition is isolated. This can be clearly seen as negative *spikes* in Figure 13.11, implying that not doing reconfiguration would have given considerably better results (e.g. at 4400 s). This is due to 4 new replicas that were to be created if clustering was done, and which would not be there if clustering was not done (i.e. there was no replica in the partition so the nodes seemingly performed local queries and thus generated no traffic). In reality, only the first node performing a query would create a local replica, while the rest would then continue to use that one, making the replica placement random, and real results would follow accordingly. Also, the minute the partition would come in contact with another one having a replica, lazy synchronization would have taken place to make sure all replicas were up to date.

In this particular scenario, the total generated network traffic during the 15000 seconds was calculated to be 36 430 MB. Synchronization costs due to reconfiguration (i.e. lazy synchronization), in the case where clustering is performed every minute, and under the assumption that data from de-allocated replicas was kept for possible future re-allocation, came to a total of 3 419 MB per hop (i.e. it is dependent on where the nearest replica is located). This is the worst case where each record to be synchronized was sent uncompressed. In reality, MDS has implemented batching, i.e. grouping all messages to be sent to one destination into one single message, in combination with compression. The real total cost for lazy synchronization, when both batching and compression are taken into consideration, would be significantly lower, with an estimated total of about 73 MB per hop (in average, about 300 KB every minute when synchronization would take place). Such a compression is possible due to the fact that the queries are plain-text, in

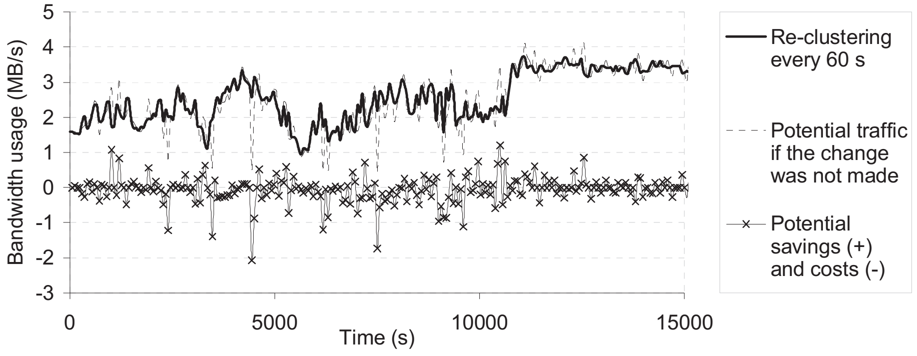


Figure 13.11: Potential bandwidth savings or costs when reconfiguration is done

addition to be similar. The standard traffic generated during the scenario by the applications, as well as eager synchronization performed immediately, do not allow for savings in such a degree, since the messages are sent separately.

The line in Figure 13.10, marked as lazy synchronization, shows the estimated cost for lazy synchronization with constant re-clustering. For comparison, the other three cases have very similar total estimated cost (ranging from 70 to 95 MB) and similarly distributed throughout the simulation, which makes the total usage generally follow the pattern of the data-only distribution.

13.7 Conclusion

Due to the dynamic nature of mobile ad-hoc networks, the availability of remotely located data might decrease substantially. Data replication provides a means for increasing data availability, but the location and number of replicas must be planned carefully. In this paper, we have presented an extensive study of different replica placement strategies for the MIDAS Data Space. Our experiments have confirmed that placement of replicas plays a big role in the amount of traffic in the network caused by usage of MDS (either direct or indirect, i.e. synchronization protocol). We could also see that having replicas on approximately 10 % of the nodes in the network gave balanced overall results when applications' access pattern was not known beforehand. These replicas also needed to be carefully placed, and different clustering algorithms have shown to be helpful in that matter, compared with arbitrary placement. We have also analyzed the possible gains and costs of performing reconfiguration of replicas in the network. The results have shown that the Traffic Controller should not blindly react to topology changes, since reconfiguration could cause a lot of undesired network traffic.

By constantly monitoring the applications' access pattern, network topology, and table sizes, the Traffic Controller can deviate from the default placement algorithm, and adapt to the situation that takes all of the factors into consideration. As part of future work, we intend to work further on optimizing the algorithm used by the Traffic Controller, as well as other protocols used in MDS. We will continue analyzing how a node's resources are affected by the overall situation in its neighborhood, as opposed to only looking at each node or link separately. Finally, we will take a deeper look into what gain can be achieved by implementing the presented idea of warm standby replicas combined with caching.

Acknowledgment

This work has been performed in the context of the MIDAS project (funded by the European Commission's 6th Framework Program, contract no. 027055), and the Ad-Hoc InfoWare project (funded by the Norwegian Research Council, IKT2010, Project No. 152929/431).

References

[1] Bellavista, P., Corradi, A., Magistretti, E., "REDMAN: A Decentralized Middleware Solution for Cooperative Replication in Dense MANETs", International Conference on Pervasive Computing and Communications Workshops, 2005

[2] Brandes, U., "A Faster Algorithm for Betweenness Centrality", Journal of Mathematical Sociology 25(2):163-177, 2001

[3] Chen, K., Nahrstedt, K., "An Integrated Data Lookup and Replication Scheme in Mobile Ad Hoc Networks", SPIE International Symposium on the Convergence of Information Technologies and Communications (ITCom 2001), Denver, Colorado, USA, 2001

[4] Clausen, T., Jacquet P., "Optimized Link State Routing Protocol (OLSR)", RFC 3626, October 2003

[5] Drugan, O., Plagemann, T., Munthe-Kaas, E., "Predicting time intervals for resource availability in MANETs", The IEEE International Workshop on Ad Hoc and Ubiquitous Computing (AHUC2006), Taichung, Taiwan, July 2006

[6] Gorman, J., "The MIDAS project: Interworking and data sharing", Interworking 2006, Santiago, Chile, January 2007

[7] Kaufman, L., Rousseeuw, P.J., "Finding Groups in Data: An Introduction to Cluster Analysis", Wiley Series in Probability and Statistics, Wiley-Interscience, 9th Edition, 1990

[8] Kitano, H., "RoboCup Rescue: a grand challenge for multi-agent systems", the International Conference on Multi-Agent Systems (ICMAS-2000), Boston, Massachusetts, USA, July 2000

[9] Munthe-Kaas, E., Johannessen, A., Pužar, M., Plagemann, T., “Information Sharing in Mobile Ad-Hoc Networks: Metadata Management in the MIDAS Dataspace”, The 10th International Conference on Mobile Data Management: Systems, Services and Middleware, Taipei, Taiwan, May 2009

[10] Newman, M.E.J., Girvan, M., “Finding and Evaluating Community Structure in Networks”, Physical Review E 69 (2), 2004

[11] Padmanabhan et al., “A survey of data replication techniques for mobile ad hoc network databases”, The VLDB Journal - The International Journal on Very Large Data Bases, v.17 n.5, p.1143-1164, August 2008

[12] Plagemann et al., “A data sharing facility for mobile ad-hoc emergency and rescue applications”, Proceedings of the First International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS 2007), Toronto, Canada, June 2007

[13] Pužar, M., Plagemann, T., “NEMAN: A network emulator for mobile ad-hoc networks”, Proceedings of the 8th International Conference on Telecommunications (ConTEL 2005), Zagreb, Croatia, June 2005

[14] Pužar, M., Skjelsvik, K.S., Plagemann, T., Munthe-Kaas, E., “Information Sharing in Mobile Ad-Hoc Networks: Evaluation of the MIDAS Data Space Prototype”, The Second International Workshop on Specialized Ad Hoc Networks and Systems (SAHNS 2009), Toronto, Canada, June 2009

[15] Reichardt, J., Bornholdt, S., “Statistical Mechanics of Community Detection”, Physical Review E 74 (1), 2006

[16] Tamori, M., Ishihara, S., Watanabe, T., Mizuno, T., “A Replica Distribution Method with Consideration of the Positions of Mobile Hosts on Wireles Ad Hoc Networks”, Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCSW’02), Vienna, Austria, 2002

[17] Yu, H., Vahdat, A., “Minimal Replication Cost for Availability”, Proceedings of the 21st ACM Symp. on Principles of Distributed Computing, pp 98–107, Monterey, CA, July 2002

[18] Zheng, J., Su, J., Lu, X., “A Clustering-Based Data Replication Algorithm in Mobile Ad Hoc Networks for Improving Data Availability”

[19] Zheng, J., Su, J., Kan, Y., Yijie, W., “Stable Neighbor Based Adaptive Replica Allocation in Mobile Ad Hoc Networks”, International Conference on Computational Science (ICCS 2004), LNCS 3036, Krakow, Poland, 2004