Universitat Autònoma de Barcelona

Departament d'Enginyeria de la Informació i de les Comunicacions

Mobile Agents in Wireless Sensor Networks to Improve the Coordination of Emergency Response Teams

Submitted to Universitat Autònoma de Barcelona in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

by Estanislao Mercadal Melià

Bellaterra, September 2013

Advisor: Dr. Joan Borrell

Professor Universitat Autònoma de Barcelona

I certify that I have read this thesis entitled "Mobile Agents in Wireless Sensor Networks to Improve the Coordination of Emergency Response Teams" and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Bellaterra, September 2013

Dr. Joan Borrell
(Advisor)

Committee:

Dr. Joan Arnedo
Dr. Frédéric Guidec
Dr. Carlos Borrego
Dr. Joaquín García Alfaro
Dr. Jordi Herrera

Program: *Doctor en Informàtica.*

Department: Departament d'Enginyeria de la Informació i de les Comunicacions.

A la meva família.

Abstract

IN this thesis, we propose an automated real-time monitoring system for victims in Mass Casualty Incidents (MCIs). New networking solutions like Delay and Disruption Tolerant Networks (DTNs) or Wireless Sensor Networks (WSNs) offer a wide array of opportunities in hostile environments where access to communications is either non-existent or broken. Due to the flexibility of WSNs, and their almost effortless field deployment, they can easily reach unexplored or unfriendly areas, monitor their surroundings and send back useful information. DTNs on their hand, can substitute a slow and high latency infrastructure based network without needing it, just using nearby resources opportunistically. In this study, we use this two technologies to create an hybrid architecture to help in the triaging of victims in emergency scenarios. The, usually, lack of a communications infrastructure, and the scarcity of resources, make this kind of scenarios a perfect place to obtain the most of DTNs and WSNs. In this thesis, we firstly present an overview of the architecture, how the two technologies are going to work together, and how they will exchange data using Mobile Agent (MA) technologies. Then, we take the explicit itinerary construction from traditional MAs and apply it to MAs working on wireless sensor nodes, with their resource restrictions and battery issues. Thirdly, we extend this itinerary structure to support WSNs clusters, fully autonomous networks with their own monitoring services, with the only limitation of not being larger than 32 nodes. This contributions are supported by tests and results which prove their feasibility and usability. Finally, we present two theoretical approaches, one to retrieve remote services in large WSNs, and another to provide access control for the nodes used in this kind of networks.

Resum

EN aquesta tesi proposem un sistema automàtic de monitorització de víctimes en grans catàstrofes. Les noves solucions d'interconnexió de dispositius, com les Xarxes Tolerants a Retards (DTNs) i les Xarxes de Sensors Sense Fils (WSNs), ofereixen un ample ventall d'oportunitats en entorns hostils on l'accés a les comunicacions és, o bé inexistent, o bé inaccessible. Gràcies a la flexibilitat de les WSNs i a la seva facilitat de desplegament, poden fàcilment accedir a zones inexplorades o de difícil accés, i monitoritzar els seus voltants per retornar informació valuosa. Les DTNs per la seva banda, poden suplir la mancança d'una xarxa de comunicacions rudimentària sense la necessitat d'aixecar una infrastructura de comunicacions, senzillament fent servir els recursos propers de manera oportunista. En aquest estudi, fem servir aquestes dues tecnologies per crear una arquitectura híbrida per ajudar al triatge de víctimes en escenaris d'emergència. L'habitual falta d'una infrastructura de comunicacions, i els escassos recursos fan d'aquest tipus d'escenari el lloc perfecte per treure tot el potencial de les DTNs i de les WSNs. En aquesta tesi primer fem una mirada general a l'arquitectura, com les dues tecnologies funcionaran juntes, i com s'intercanviaran dades fent servir tecnologies d'Agents Mòbils (MA). Després, agafarem la construcció d'itinerari explícit dels MAs tradicionals i l'aplicarem als MAs funcionant sobre WSNs, amb les seves restriccions de còmput i de bateria. En tercer lloc, extendrem aquesta estructura d'itinerari explícit per funcionar amb *clústers* de WSNs, xarxes totalment autònomes amb els seus serveis de monitorització, amb l'única limitació de no poder ser més grans de 32 nodes. Tot això recolzat per proves i resultats que demostren la seva viabilitat i utilitat. Per acabar, presentem dues propostes teòriques, una per accedir i recuperar serveis remots en WSNs grans, i una altra per proporcionar control d'accés als nodes en aquest tipus de xarxes.

Acknowledgments

THIS thesis would not have been possible without the help and support from very valuable people. It is difficult to choose somebody to start with, but I will start in the most traditional way.

I thank all my family, my parents, my brothers, my grandparents, my uncles and aunts and my cousins for their support, pressure and trust throghout all my life and education, this pages are truly full of you. *Moltes gràcies família*!

To my friends, both from Menorca and those I made during my years in Barcelona, friends that without them I would not have been capable of completing this thesis, friends that helped me in my best and in my worst moments. In just two words, real friends. Thank you all!

Thanks to all the people at MISL, specially to Ilias Tsompanidis and Jason Quinlan who welcomed and helped me during my stay in Cork. I will never forget our discussions about English pronunciation, *"a cheap ship ships sheep eating chips"* and my willingness to learn both Irish and Greek, I still remember some words! ... a bit. I really hope we will meet again! Ευχαριστώ, Ilias; *Go raibh maith agat*, Jason. And without leaving Cork, I don't have enough words to thank Cormac Sreenan, my supervisor during my stay there, for his generous and stimulating guidance, and for his disinterested help in many stages of my research and writing.

I would also like to thank all my colleagues in dEIC, my home department, for the working and, mostly, non-working moments we had together, they truly helped me to carry on with the research and with this thesis. You will forgive me here if I don't write any names, I don't want to miss anybody nor put anybody ahead, you all are awesome! Also, I think that I speak in the name of my family when I say thank you all for buying the typical Menorcan *sobrassada* and support the family businness.

I cannot forget the person who brought me to the department, Sergi Robles, thank you for believing in me and for the first years of my tuition and the great times we spent together talking about everything and, just briefly, computer networks.

Of course I will not forget my thesis supervisor, Joan Borrell. He really helped me a lot to get along with this new life as a researcher. I would like to thank not only all his efforts to take my work to fruition, but also all he has done to make me feel like at home.

And finally, thank my personal muse for her inspiration these last days, without her guidance, this thesis would be incomplete. Lots of love.

This work has been partially supported by the Spanish Ministry of Science and Innovation through the project TIN2010-15764 and by the Catalan AGAUR 2009SGR-1224 project. I would also like to thank this institutions for their funding, but also want to ask them to reconsider their last research and education budgets, to think about the future of the country. A country with a poor education and not committed with knowledge will never be a rich and prosperous country.

Estanislao Mercadal Melià

Barcelona, September 2013

Contents

List of Tables

List of Figures

Part I

1

Introduction

I'm on my way home after a long walk. Today I've taken the long route, not because I wanted to, but because the stress monitor in my glasses told me my body needed it. I stop at my home's front door and I can see the stress is far lower than one hour before, much better. My shoes had already sent my walk summary to my glasses, pace, kilometers, route, ... I would never have thought you can spend that many calories just by walking! My watch sent my heartrate, and a list of people I have crossed, with links to their social network profiles, nobody interesting. I enter the kitchen and take the glass of isotonic drink my fridge poured and I accept the shopping cart prepared by my fridge and my pantry. In my way to the bathroom I can feel the floor warming up under my feet, the house knows I'm here and where I'm coming from and turns on the heating in my usual route after a day like this. I enter the bathroom and find the tub already filled with warm and relaxing water, ready for me to smoothly slip into it.

THE previous situation is not, of course, the present day, but it also is not a very distant future. As time passes by the Internet is becoming more and more ubiquitous, reaching unimaginable objects, places

and devices. One simply has to look twenty years back, with the Internet just in the beginnings of its adulthood. At that time, the early 90s, it left the academia and made its first appearances in people's homes, connecting them to an unknown and unexplored global network. Not many years later this stranger became a regular inhabitant in our homes, showing us a new widely communicated world to explore, to invest and to investigate, but it was still enclosed in computers, and users had to sit down and consciously connect to the Internet. Computers were, at that time, the only window to look at that young and untamed world. In this last years it moved to our pockets, to our TVs, to our watches, ..., every day we see things that we had never thought they will be connected to the Internet, and they do so transparently to the end user, some even not knowing they are using the Internet for a handful of daily tasks). What to expect from the near future? The Internet of Things was a term proposed in 1999 by Kevin Ashton [Ash09], and what it implies is precisely what was depicted in the introductory story, a world where every device is connected to the Internet, and working transparently for us.

To take advantage of all the possibilities the Internet of Things offers its devices must have sensors with which receive outside inputs to process them and respond adequately. What is more, they should create networks of nearby sensor equipped devices and share their inputs to generate an adequate response just in time for the user to use it. In a household environment this has, probably, no difficulties, as access to a permanent network connection and permanent power source are assured. In not so closed and safe scenarios, though, one has to think of alternate ways to move data to its final destination without draining too much power from sensing devices, i.e. avoiding expensive, in every possible definition, data transport mediums (WiFi or mobile data).

Wireless Sensor Networks (WSNs) is the term coined for this kind of independent networks of small devices which may or may not have a direct exit to a larger network, but are obliged to wait patiently until the right time for communication comes. Minimizing battery and radio usage, while maintaining a good update policy is key in this kind of scenarios where access to updated information is a must or greatly improves the subsequent response. The situation depicted in the above example is not a critical one, but having the right information helps actuators to provide a more accurate response or, on the other hand, in case of an accident it may be of great use to transmit this data as fast as possible to the nearest medical resource.

Looking at how the world is becoming more and more connected, we strongly believe that sensor equipped devices will be an active part of our cities, making our daily live easier and more comfortable. WSNs will be, of course, critical elements of this *smart city*, by collecting environmental data and using our mobile devices as carriers for their collected information, avoiding the use of a permanent wireless connection which will rapidly drain device's battery.

The contributions of this thesis are focused in finding solutions for the key issues in data distribution inside WSNs and find a compromise between battery usage and data freshness at end points. The starting point of our contributions is an existing application of our group (Mobile Agent Electronic Triage Tag (MAETT) [MRMCC09]), which uses traditional Mobile Agents (MAs) to manage victims in Mass Casualty Incidents (MCIs). This application uses a form of communication not backed by end-to-end connectivity, but on a point-to-point communication approach, choosing from their connected devices which is the best to send its information, if it exists. This form of communication is similar to what we know today as Delay and Disruption Tolerant Networks (DTNs), where nodes follow a carry and forward approach, storing their data for as long as they find a suitable node to send their information.

Our proposal is to use MAs to design and build a complex application with WSNs. The difference between our MAs and those used in the previous work is that they are executed in the WSNs and have to deal with battery life and computing performance issues. Up to now, all applications build with MAs in WSNs are mere demonstrations of the possibility of using this technologies together. On the other hand, we want to demonstrate that MAs in WSNs are a real combination of technologies to be used in complex situations.

1.1 Objectives

Broadly, the objective of this thesis is to provide a complete MCI victims monitoring system not relying in traditional infrastructure supported networks.

Concretely, the objectives are:

1. Design and develop a hybrid architecture of DTNs and WSNs to continuously monitor victims and effectively communicate monitored data in a MCI introducing WSNs and MAs inside these. WSNs
2. Extend this architecture to perform reliably and efficiently regardless the number of victims in the MCI
3. Design and develop a system to locate qualified medical personnel in an affected area
4. Design a protection system for the information gathered by the monitoring application and prevent its unauthorized access

The combination of DTNs, WSNs and MAs offer new triaging systems, which end up with more effective victim rescue planning and a better use of the available human resources. Our aim was to study how these technologies can change, for better, the way lives are rescued in big emergencies. The objectives have been achieved by using low power sensor nodes forming WSNs to monitor victims, mobile agents to read, aggregate, share and carry sensed data, and DTNs to carry this data to an Emergency Coordination Center (ECC).

1.2 Results

At the time of this thesis writing, we have built a complete application to efficiently manage active rescue teams in WSNs, all of this using what modern information technologies offer. We designed a use case scenario where sensor nodes monitor MCI victims, information is collected by rescue teams and opportunistically sent to an ECC which can then plan a more efficient rescuing scheme, always with the most updated information about the health state of the victims.

Moreover, we have built the application to be easily extensible and fault tolerant, providing means to add and remove nodes and groups of nodes most of the times with little or no impact to the application's performance.

We also designed and tested an algorithm to search and retrieve remote and mobile services in a large WSN, specialized medics in our case. This algorithm has been tested successfully in small networks.

Finally, we provided a theoretical approach to security with mobile agents in

low power sensor nodes using an efficient algorithm that provides reasonable access control protection without affecting in a considerable manner system's performance.

Positive results during the research have been published in both national and international conferences and journals:

- Estanislao Mercadal, Carlos Vidueira, Cormac J. Sreenan, and Joan Borrell. Improving the dynamism of mobile agent applications in wireless sensor networks through separate itineraries. *Computer Communications*, 36(9):1011 – 1023, 2013. ISSN 0140-3664.
  URL `http://dx.doi.org/10.1016/j.comcom.2012.09.017`. [MVSB13]
- Estanislao Mercadal, Sergi Robles, Ramon Martí, Cormac J Sreenan, and Joan Borrell. Double multiagent architecture for dynamic triage of victims in emergency scenarios. *Progress in Artificial Intelligence*, 1(2): 183–191, 2012. [MRM+12]
- Estanislao Mercadal, Guillermo Navarro-Arribas, Simon N Foley, and Joan Borrell. Towards efficient access control in a mobile agent based wireless sensor network. In *7th International Conference on Risk and Security of Internet and Systems (CRiSIS)*, 2012, pages 1–4. IEEE, 2012. [MNAFB12]

1.3 Document Layout

This thesis is structured in four separated parts. In the first one required concepts for the correct understanding of the thesis are presented, separated in three main blocks, **Introduction**, **State of the Art** and **Heterogeneous Multiagent Architecture for Emergency Management**. They mainly develop what we have sought writing this thesis, detail the technologies and their state prior to the publication of this thesis and describes the general scenario used to test our research.

The second part, our real contribution, is structured in four independent chapters: **Mass Casualty Incident Management with Mobile Agents and Wireless Sensor Networks**, **Clusters in WSNs using Mobile Agents**, **Access Control in WSNs using Mobile Agents** and **Retrieval of Remote Moving Data in WSNs with Mobile Agents**, each presenting a novel technique demonstrating how existing networking technologies can be

used to solve various problems, all of them exemplified using a handy MCI scenario where this technologies can deploy their potential.

Part three, exposes the combined conclusions of the thesis and presents some future improvements to the presented architectures and algorithms which will make the contributions more effective, easy to use and efficient.

Finally, the fourth part, is exclusively dedicated to the bibliography and the acronyms used in this thesis.

2

State of the Art

THIS chapter introduces the reader to emergency scenarios in Mass Casualty Incidents (MCIs), we apply our research results to this kind of scenarios as they can be greatly benefited of it. We also describe the key technologies used in this thesis, namely Wireless Sensor Networks (WSNs), Delay and Disruption Tolerant Networks (DTNs), and Mobile Agents (MAs), which are used to automate important parts of victim management in emergency scenarios. The reader will also find the state of the art of this technologies, each in an independent section of the chapter. We begin by describing emergency scenarios and the state of the art of the administration of their victims. In the second section the low power requirements of the WSNs and the mobility of some of their nodes to further improve their power consumption is described. The third section is devoted to DTNs, a special type of network where end-to-end connectivity is not assured. Finally, the fourth section presents MAs, a programming paradigm that allows code and data to be moved altogether between nodes in a network.

All this technologies are put to work together to improve a previously existing victim triage system in MCIs, known as Mobile Agent Electronic Triage Tag (MAETT). MAETT uses an early form of DTN (see 2.1.2) to transparently send victim's triage data to the designated Emergency Coordination Center

(ECC). We use this system as a base and add MA powered WSNs to provide real time victim triage, as well as other improvements described throughout this thesis.

2.1 Emergency management

After a big catastrophe, a hurricane, a terrorist attack or a nuclear meltdown, for instance, reacting immediately and adequately is paramount, as is the correct administration of available personnel and resources. In most cases, communication networks are disrupted and nonfunctional, increasing the difficulty of the planning.

The affected zone is divided in 3 different zones depending on their situation or function in the emergency. **Zone 0** is where the disaster took place, thus, where the catastrophe's victims and injured are. **Zone 1** is where all the emergency's medical staff gathers. It is also where the field hospital is located and where rescue personnel gathers. Finally, **Zone 2** is composed of all other medical effectives around the globe available to collaborate.

Effectives are coordinated by the ECC, located in zone 1, to ensure the proper treatment and evacuation of affected victims. Triaging members scour zone 0 randomly searching for injured victims and tagging them following an standard protocol, e.g. Simple Triage And Rapid Treatment (START) [Sup84], but not treating them. At the same time, various specialized medical teams equipped with medical supplies, search for already triaged victims and treat them, be it in-place or in an ambulance, moving them to the field hospital. If the ECC knows of a particular zone with critical victims, it can directly send a medical team to that particular area, speeding up the treatment and evacuation of those victims.

The sorting of victims during the first moments of a MCI aftermath is one of the most important tasks, and is a crucial aspect for the efficiency of the rescue teams. Existing protocols like START or Manchester Triage System (MTS) [MJMW06] classify victims depending on the severity of its injuries using a simple color code system.

Traditional identification methods use a standard physical cardboard tag which is placed on the victim, usually around its neck. This tag contains, generally, the color obtained with the classifying algorithm and other useful

data about the health of the victim. Afterwards, medical personnel use this tag to allocate resources more efficiently.

Although this approach is widely accepted and used [IMM+06, NS10, NG99], there is room for improvement and many electronic systems have been proposed to address what are considered the two main issues of victim triage systems:

1. The physical nature of the tag placed on victims' bodies, which hinders their localization due to its dependence on the visual skills of the medical personnel and on direct eye contact. As an example, what would happen if wind moved the tag behind the victim's body?

2. The task of sorting and locating victims for further assistance and transportation are decoupled. Tagged victims' tracing to verify their collection status is not possible, as is not the finding of a high critical victim concentration in the emergency area.

Proposed electronic systems to solve these problems range from quick infrastructure deployments [DR07] to barcode or Radio Frequency Identification (RFID) based solutions [ISOF06]. All of them try to solve the main issues of emergency management, but fail to provide a feasible low-cost solution to solve them, and especially to early resource provision.

As has been done in many other scenarios where distribution and coordination are of capital importance, agent technology has also been used for emergency management [FL05], though agents' purpose is not that of triaging. Nonetheless, the introduction of agents in victim triaging meant an important advance in emergency management systems, and the number of applications in related areas is sensitively growing [Hen06].

2.1.1 Mobile Agent Electronic Triage Tag

MAETT [MRMCC09] went a step further and applied MAs and an early form of DTN to the actual triaging system. In their proposal they add a contact-free, low-range RFID device with a unique victim identifier, and a handheld device with a Global Positioning System (GPS) receiver, a touch screen, a RFID reader and wireless communications. This device is affixed to the high visibility vest worn by the field personnel.

MAETT's foundations rely on mobile agent technology [CAM07], which allows information to be directly transported from terminal to neighboring terminal regardless of the status of the rest of the network at that particular time. GPS equipped handheld devices worn by medical personnel run an execution environment for agents, known as the platform, JADE in our case, where mobile agents can be created, executed and forwarded to other terminals. Depending on the available information on their neighbors, agents themselves, not the platform, nor the bearer, decide the route to follow to efficiently deliver the collected information.

The main actors of MAETT are the victims, the triage personnel, and the rescue teams (See Figure 2.1). Due to the nature of a MCI victims are scattered over an arbitrarily large area, triage personnel scour this area searching and triaging them according to standard methods (START, MTS, ...). The result of the triage algorithm is written in a physical tag and placed visibly on the victim. Finally, the rescue teams collect all the victims prioritizing them depending on triage results. All this operations are coordinated by the ECC.

Physical cardboard tags placed on victims have an integrated RFID and, at the same time of the triage, an agent is created containing the information in the tag, plus the GPS position of the victim and the RFID of the tag. All this information is later used in the ECC to optimize rescue teams' routes.

When triage personnel leave the ECC, they have an estimation on when they will get back, the Time to Return (TTR). Agents use this TTR, available in triage personnel handhelds, to decide whether it is worth to migrate from one handheld to another, an agent will only change its host if the TTR is smaller is the destination. This is to make sure that a migration will never increase the time for an agent to reach the ECC. Eventually, when agents arrive at their destination, the ECC, rescue teams are sent to rescue victims with a detailed schedule of the route based on victim's GPS position and their medical condition.

In Chapter 4 we improve this construction by adding small wireless devices equipped with sensors to victims. This small devices build up a Wireless Network and is prepared to do a dynamic triage of emergency victims.

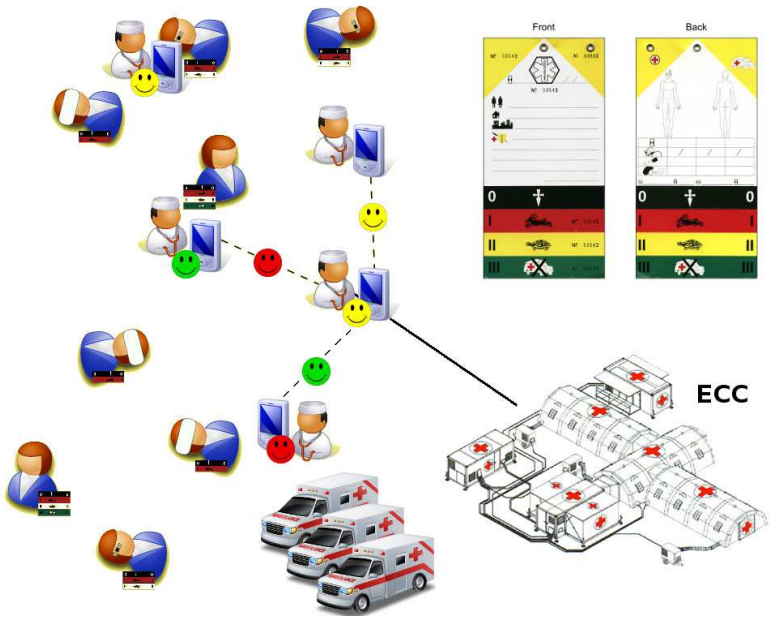


Figure 2.1: MAETT basic scheme with the traditional cardboard tag.

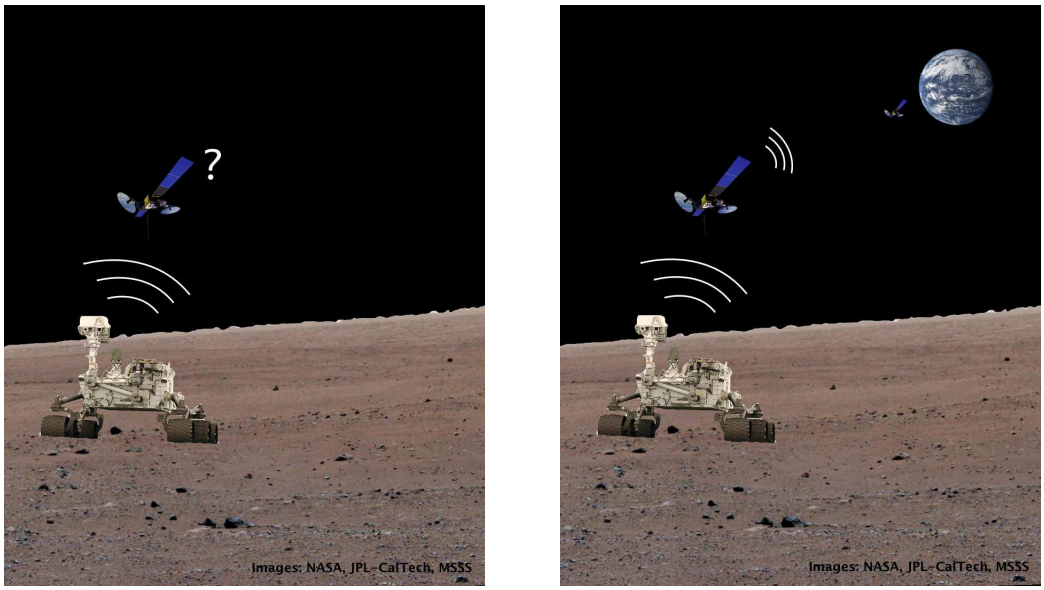
2.1.2 Delay and Disruption Tolerant Networks

DTNs are a special type of networks not governed by the classical end-to-end paradigm found in TCP/IP networks. In this particular networks, nodes can be completely isolated for an extended period of time until they can effectively dump their data to the next node of the route. Protocols suited for Mobile Ad-hoc NETworks (MANETs) or ad-hoc networks, such as AODV [PR99] or DSR [JMB+01], are useless for this kind of networks due to its low tolerance to latencies, delays and broken links.

As a network of this type is always changing, its complete knowledge is not possible and routing protocols designed for DTNs should rely on heuristics and will be far from optimal. Despite of that, a bunch of routing protocols for DTNs emerged since the apparition of these networks, being epidemic routing [VB+00] the most simple one, which replicates the message whenever possible, with no additional routing logic. Other routing algorithms like PRoPHET [LDS04], MaxProp [BGJL06], RAPID [BLV07] or Spray and Wait [SPR05] considerably reduce the number of copied or relayed messages by applying additional logic to the node selection.

To illustrate the importance of DTNs in today's and future communications

just state that NASA is considering them for its future space operations [nas]. On the other hand, in MAETT they are used to opportunistically send victim's data to the best candidate device, with the objective of reducing the time needed for victim's information to reach the ECC.



(a) (b)

Figure 2.2: Author's illustration of DTN for space operations.

2.1.3 Mobile Agents

Software agents appeared for the first time in the literature in the late 1970s: "*... is a computational agent which has a mail address and a behavior. Actors communicate by message-passing and carry out their actions concurrently ...*" [Hew77]. Their mobile counterparts firstly appeared in the second half of the 1990s when proposals like Java Aglets [LOKK97], Java to Go [LM96], Mobile Objects and Agents [MLC98], MOLE [SBH96] and Telescript [Whi99] appeared in the press.

The MA paradigm implies that application code and data live together in a single software entity. As a whole it can be moved between nodes of a network and regenerates its execution state and data in the new node (strong migration), or start anew (weak migration). This new paradigm presents a new situation for network administrators: the job previously done in a

centralized manner or using more than one application can now be done with a single moving software unit!

Since the apparition of Concordia [WPW+97] mobile agents moved through nodes of the network in an implicit manner, i.e. the next node for the agent to migrate was hardcoded into the agent's code. Any modification of the network and agents' code should be changed to reflect the changes in the itinerary. Concordia introduced a separate structure holding the information of the locations to be visited by the agent. Apart from migration instructions, this new structure, included specific code and data for each visited host.

Separate itineraries were first sequential, the agent just followed the list of nodes to visit one after the other, in the order specified by the programmer. Flexible itineraries were introduced in the late 1990s [SRM+98] and allowed agents to take decisions about their travel plan at runtime depending on the type of the entry in the itinerary: the *sequence*, where only one possible destination is offered to the agent, similar to the itineraries presented by Concordia; the *alternative*, where some different predefined hosts are offered to the agent and one has to be chosen; and the *set*, where all the hosts from a predefined set must be visited by the agent.

Another step forward in the topic of mobile agent itineraries, a proposal from 1998 [GRB08] defined and protected separate itineraries for *free-roaming agents*, involving the discovery of the location of one or more destination nodes at runtime.

2.1.4 Mobile Agent Middlewares

In the first years of the second coming of agents in the 1990s a non profit organization was founded in Switzerland with the intention of defining a full set of standards for the building and communication of agents [ON98]. From that moment many standards where proposed and adopted by many platforms [BPR99, HKH09, GCB06, VGB08], being Agent Management [FIP04] and Agent Communication Language [Fip02] the most widely used. In 2005 the organization dissolved and became an IEEE standards committee.

For the work presented in this thesis the platform of importance is JADE [BPR99], first appeared in 1999 and still actively developed [jad].

2.1.5 Mobile Agents in DTNs

MA, with their ability to move code and data autonomously are great candidates to solve routing problems in DTNs. Their ability to execute code in the platform they are on, and to reason which is the most suitable next node to jump to depending on the environment, they can solve the routing problems found in DTNs easily and without the need of charging every host with different routing algorithms.

In MAETT, for example, victim's data is stored in MAs and they are used to route this data in the constrained environment that is an emergency scenario until reaching the ECC (see Section 2.1.1). Every time a victim is triaged a MA is created containing victim's data, being its only objective to reach an ECC in the smaller amount of time to improve the rescue plan and save the maximum amount of lives as possible, being the agents themselves the ones taking the decisions.

2.2 Wireless Sensor Networks

Wireless sensor networks (WSN) are an actively researched technology, with around 19000 published scientific papers in 2012[1], a very similar number to those published in the topic of "Near Field Communications", due to its potential to bring computing to remote or disconnected areas, where ordinary computing devices are not prepared to work. Long battery life and low powered wireless communications are a must in this type of networks, where its nodes can be unattended for long periods of time collecting data of its surroundings.

They are a specific type of *ad-hoc* network consisting of hardware sensors collecting particular measures, i.e. temperature or blood pressure, and processing elements, which collect these measures for further processing.

Devices forming this kind of networks are usually small, equipped with low power processors, a bunch of sensors and are powered on batteries, depending on the application lasting for months.

Traditional WSNs also require a special node, usually bigger and connected

[1]Google Scholar search for "Wireless Sensor Networks" and "Near Field Communications", Last Accessed: 2013-04-05.

to a larger power supply and to a network backbone known as the *sink*. The other nodes of the network send their collected data to this special node, which either makes the calculations itself and waits until the results are collected, or sends the gathered data through a traditional network to a place where it will be processed offline by the interested people.

On top of the processing and sensing hardware lies the operating system. The most used ones are TinyOS [HSW+00] and Contiki [DGV04]. TinyOS was released in the year 2000 by a co-operation between the University of California, Berkeley, Intel Research and Crossbow Technology. Nowadays it has grown and is an international consortium known as the TinyOS Alliance. It runs on top of a large number of hardware devices such us Imote2, IRIS, MicaZ, Mica2, Mulle, TMote Sky (Telos rev. B), UCMote Mini, even on Lego Mindstorms NXT, among others. TinyOS applications are written in nesC, a component-based, event-driven extension of the C programming language and simulated using TOSSIM, the TinyOS simulator, which provides all the interfaces needed to simulate an application as if it was running on a real mote. The last stable release is 2.1.2 from August, 2012 [tin].

Contiki on its hand was first released in 2003 by Adam Dunkels and is now being actively developed by a large group of developers including Cisco, RWTH Aachen University or Oxford University. As TinyOS, Contiki runs on top of many hardware platforms such us MicaZ, TMote Sky (Telos rev. B), Wismote and many others. Contiki applications are simulated with Cooja, its network simulator, that accepts three classes of nodes: emulated, where all the node hardware is emulated; Cooja nodes, where the code is compiled for the simulation host; and Java nodes, where the node has to be reimplemented as a Java class. A simulation can contain a mixture of these three classes. The last stable release is 2.6 from July, 2012 [con].

2.2.1 Mobile Agents in WSNs

Some approaches to bring mobile agents to Wireless Sensor Networks appeared in the first decade of the 21st century. Middleware proposals as ActorNet [KSMA06], In-motes [GB06], Agilla [FRL09], MAPS [AFGG11] and some others, emerged in the research community. Of all of them only the aforementioned, according to their authors, ended up with an actual implementation for one or another hardware platform and operating system. ActorNet and In-motes for TinyOS over Mica2 nodes, MAPS for Sun SPOTs

and Agilla for TinyOS over a variety of hardware platforms, mainly Telos rev. B and Mica[X]. Refer to Table 1 for a full reference.

Among the different mobile agent middlewares proposed for WSNs, the first one deployed inside real WSNs is Agilla [FRL05, FRL09].

The Agilla middleware

Agilla provides a programming model in which applications consist of evolving communities of agents that share a WSN. Agents can dynamically enter and exit the WSN, can autonomously clone and migrate themselves in response to environmental changes, and can maintain a global coordination through a tuple space, a type of shared memory accessed via pattern-matching that enables a decoupled style of communication. The size of the tuple space is up to 48 bytes in each node. Agilla was implemented on top of TinyOS WSN operating system [HSW+00], and experimentally evaluated on several real WSNs, for instance those consisting of TelosB [PSC05] nodes. A basic Agilla installation in a TelosB node takes up 3866 bytes out of 10KB of RAM and 45308 default bytes out of 48KB of ROM. See Figure 2.3 for an illustration of Agilla's architecture.

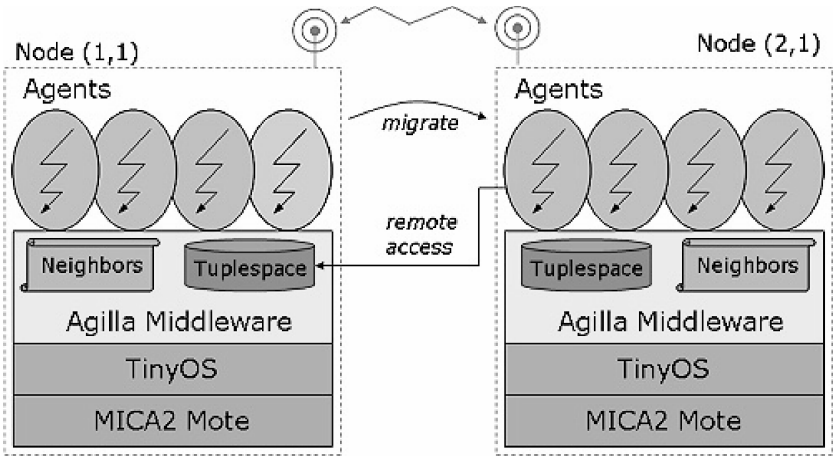


Figure 2.3: Agilla Middleware Components
(from `http://mobilab.cse.wustl.edu/projects/agilla/`).

Every Agilla node supports multiple mobile agents able to move or clone across other nodes while carrying their state. To facilitate agent interactions, every Agilla installation provides the node with two data structures, a

Agent Framework [SLO05]

Platforms:	Mica2dot
Prog. language:	Maté [LC02] TinyScript
Coordination:	Shared memory, network messages
Applications:	Global data collection Gradient search Event tracking

ActorNet [KSMA06]

Platforms:	Mica2
Prog. language:	High level functional-oriented
Coordination:	Shared memory, network messages
Applications:	Gradient search

In-Motes [GB06]

Platforms:	Mica2dot
Prog. language:	Micro-programming
Coordination:	Tuple spaces, agent facilitators
Applications:	Data gathering

WISEMAN [GVVL06, GVCL10]

Platforms:	MicaZ
Prog. language:	Text-based codes
Coordination:	Local (node) variables
Applications:	Early forest fire detection

Agilla [FRL05, FRL09]

Platforms:	TelosB, Mica2, MicaZ and Tyndall 25mm
Prog. language:	Micro-programming
Coordination:	Tuple spaces [Gel85]
Applications:	Fire detection and tracking Monitoring cargo containers Navigation in a dynamic environment

MAPS [AFGG11]

Platforms:	Sun SPOT
Prog. language:	Java
Coordination:	Network messages
Applications:	Remote sensor monitoring

Table 1: Reference of Mobile Agent Platforms for Wireless Sensor Networks.

neighbor list and a tuple space, a sort of shared memory accessed via pattern-matching queries that enables a decoupled style of communication. Agilla also provides specialized reaction primitives enabling agents to efficiently respond to changing states. Prior Agilla versions addressed WSN nodes by their location in a grid structure, this restriction was removed in version 3.0.

Concerning memory, the Agilla middleware provides three different data storage constructions:

The tuple space is a shared memory space where data is structured as tuples that are accessed via pattern-matching queries or *reactions*. It is used primarily for communication between agents, either coexisting in the same node or not.

The stack is a simple LIFO queue that provides only two operations `push` and `pop`. It is fundamentally used to store application runtime variables and instruction return values. In a default Agilla installation it can store up to 105 bytes.

The heap is a random-access storage area that allows agents to store, in a default Agilla installation, 20 variables of 16-bit each. It is a random access memory construction accessed with the `setvar` and `getvar` instructions.

A `reaction` is a method that makes agents respond to the presence of a certain tuple in the tuple space, preventing them from performing continuous polls. They, reactions, consist of a tuple space template to react to, a label to the callback function, and a block of code. Once registered (`regrxn`), they notify the agent if a tuple matching the template is pushed into the tuple space, and provides a jump position for the agent code to jump to and respond to the tuple.

Regarding itineraries of MAs in WSNs, optimization of energy consumption in their planning is paramount. As the problem of finding optimal itineraries in WSNs is NP-hard [WRB+04b], a lot of research has been devoted to this problem, surveyed in [CYK+11]. Different heuristics have been proposed, from the simplest ones [QW01], based on genetic algorithms [MFV+06, CCH+11], to more elaborated ones [CYK+11]. Multiple mobile agents' itinerary planning is also considered in recent proposals [WCKC11, CYK+11] to allow the scalability of the solutions to large WSNs.

2.3 Summary and Conclusions

Albeit the middlewares and the research done in the mobile agents in WSNs topic, they still need a killer application to move them out of their cradle. The applications used to show the goodnesses of the technology don't really unveil the full potential of mobile agents, and stick with simple tasks easily done without mobile agents and where their benefit is hardly noticeable.

This is not the case of our application proposal, as the reader will see through the pages of this thesis, ours is a complex application using several novel technologies to accomplish an important task, which is the automation of the classification of victims in MCIs.

We believe the application has enough potential to become the launching platform for future WSN applications that will help both our daily and professional lives.

Part II

3

Heterogeneous Multiagent Architecture for Emergency Management

BEFORE entering in detail to the contributions of this thesis we will take a quick glance to how all of them are supposed to work together.

Agilla and JADE, despite being two very different agent technologies, can coexist and share information to build a more complex agent system. Albeit agent migration between both platforms is not possible from the very beginning, little changes in Agilla's code make this cooperation possible by embedding an Agilla agent into a JADE agent.

The first step of our contribution consists on using Agilla to continuously monitor Mass Casualty Incident (MCI) victims inside Wireless Sensor Networks (WSNs) and use JADE agents to carry the monitored data to the Emergency Coordination Center (ECC), introducing dynamism to Mobile Agent Electronic Triage Tag (MAETT). We take advantage of the communication between the two technologies to share victims' information and route details, thus improving the efficiency of the triaging system.

We extend MAETT's scheme (Figure 2.1. See Section 2.1.1 for a detailed

description of MAETT) and add a wireless human monitoring device, similar to the one depicted in Figure 3.1a, a TelosB compatible node manufactured by Maxfor (`http://www.maxfor.co.kr`), that creates a WSN with its neighboring nodes, which we later use to dynamically update the medical status of every victim.

Medical personnel in the affected area also carry a WSN node to enable their communication with the victim's network. In this case the node is attached to a handheld running JADE, working as a Delay and Disruption Tolerant Network (DTN) node (Figure 3.1b).

(a) Example of wireless human monitoring device.

(b) N810 handheld with an attached WSN node

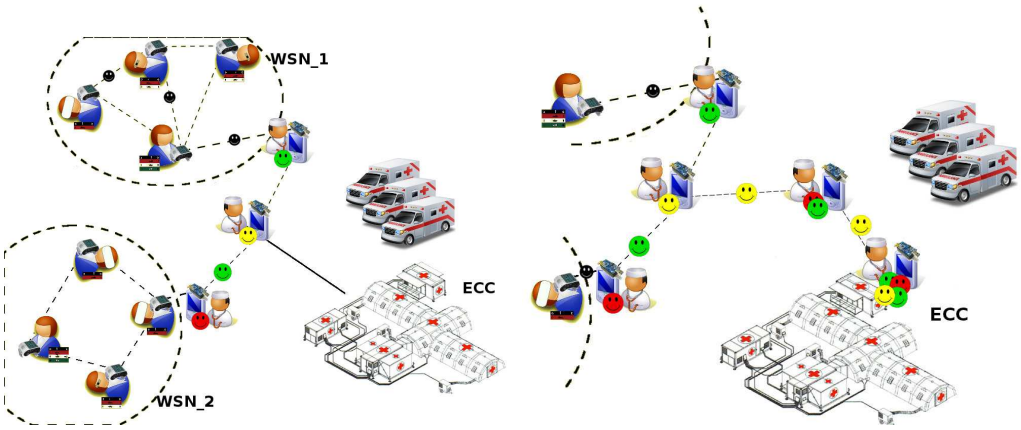
Figure 3.1: Two types of sensor nodes used in our architecture.

Apart from the physical cardboard tag, triage personnel also place a wireless capable monitoring node running Mobile Agents (MAs) (Figure 3.2a to every triaged victim to continuously monitor their health status, computing a health summary following an algorithm similar to Simple Triage And Rapid Treatment (START) or Manchester Triage System (MTS). As nearby victims are both paper and electronically tagged, neighboring nodes belonging to the same triage member wirelessly connect, creating a growing WSN of victims.

When every victim in the vicinity is tagged, and thus, every body sensor is monitoring its own victim, the triage personnel member ends the creation of his WSN. At the same time, the triage personnel handheld starts the calculation of an itinerary through the newly created WSN, which will be used by a MA to fetch the updates of every victim's medical condition.

To increase the probability of the collected information being correctly sent

from a node in the WSN to a handheld device carried by a medical personnel member, all status updates are shared by a mobile agent to the other nodes of the WSN. This information waits for a passing by handheld, carried by any medical personnel member in the emergency. A JADE mobile agent is finally created in the handheld, containing the new health summaries of the victims in the emergency, which is routed to the ECC applying the strategies used in MAETT (Figure 3.2b).



(a) MAETT with continuously monitored victims.

(b) Detail of the return path to the ECC.

Figure 3.2: Our extension of MAETT.

3.1 WSN - DTN interface

For all the application to share data between medics, to move data to the ECC, in other words, to be of use, we need to convince JADE and Agilla agents to work together. We had to modify Agilla's AgentInjector in the handheld to obtain an instance of the running JADE platform or switch on the platform if not running, this way we ensure that every received agent in the handheld will generate a JADE agent.

Agents migrating to the handheld, the one to be encapsulated into a JADE agent, needs a special piece of code to enable the communication with the Injector at the other side of the USB interface. This code allows data in the wireless node to be moved to a special address pertaining to the injector in the handheld device.

Our test platform, a Nokia N810 with MaemoOS Diablo (5.2008.43.7) did not come with appropriate drivers to work with our TelosB motes. Thankfully, MaemoOS is a Linux kernel based open source OS and we were able to recompile the stock kernel to include the required driver. Moreover, albeit the handheld device supports *host mode USB*, not every USB cable has the required pin configuration to automatically notify their host of the requirement. Fortunately, a simple command on our test platform was sufficient to enable it. At that point the system recognized a new USB device, but complained about being unable to access it. The problem was related to the platform not feeding enough power to the USB device. Writing a special *udev* rule, specific for our type of sensor nodes solved the problem and, finally, the we had our test platform completely functional.

3.2 Traveling the WSN

Visiting every node in the newly created WSN to share data between nodes, preferably avoiding repetition and optimizing the number of hops, is thoroughly studied in graph theory, also known as the Traveling Salesman Problem, and known to be NP-Complete [Pap77], even in the Euclidean plane.

In our case, covering all the nodes of a WSN can be seen as specific case of the problem, with the particularity of also trying to optimize energy consumption, which happens to also be NP-Complete [Mas05, WRB+04a].

Genetic algorithms have been used to solve NP-Complete problems since the beginning of the 90's [JS89] and, particularly, to efficiently solve the specific case of the Traveling Salesman Problem [Bra91, SRJB03, LnKM+99].

As we use a time and power restricted battery powered handheld device, it is not feasible to calculate the optimal route for a problems of this kind. Thus, we use a well studied genetic algorithm approximation [Mas05], which is proved to offer good enough solutions. Albeit not being optimal, the returned solution is a satisfactory route, both in computation and traveling times.

The genetic algorithm starts by creating an initial population of random paths to cover the network. A new generation is started by selecting only the paths whose number of hops to cover the network is smaller or equal to the remaining hops to meet the threshold (maximum number of nodes per

path) are selected. Thereafter, x individuals are selected to be crossed, be it a subset of their paths or two entire individuals. Finally, after a limited number of crossings and new generations, all solutions are evaluated and sorted in ascending order, from those with the smaller number of hops to those with the highest, and the best one is selected.

This was our first approach and worked well with open field networks and fully connected networks, without obstacles. When this premise is not satisfied, the cost of generating valid individuals is too high to be used in a resource constrained environment, both in time and power, as is our handheld device. Indeed, in scenarios with broken links, it is very difficult to find valuable individuals after every iteration, due to the high cost of finding a valid route through the graph in the crossing and mutation phases of the genetic algorithm.

That is why we finally opted for a simple Depth-First Search (DFS) algorithm [IPP+10], which computes the spanning tree of a given graph. In our case, the returned tree is the path the Agilla MA will use to visit all the nodes of the network.

It is worth noting that we can calculate a path using the DFS algorithm in the handheld due to the following facts:

1. The handheld has the topology of the WSN, saved by the triage personnel while tagging victims. When the triage member closes the network, the device starts calculating a network covering path and injects a MA with the final route into the WSN.

2. The number of sensing nodes is limited. Both, to speed up the route calculation process, and to lighten the medical personnel bags. The weight of a health monitoring sensor, albeit it may seem negligible is around 70g including two 1.5V AA batteries, and carrying more than 2Kg (∼30 nodes) may hamper the medical personnel.

Furthermore, to restrict the calculated itinerary to our concrete needs we introduced three limitations to the DFS algorithm to return a better solution for our network:

1. The initial node is the last victim triaged by a personnel member. This prevents the triage member of having to move to the best possible starting node, but inject the node where he is.

2. The itinerary does not need to be cyclic. The last and the first node

of the network do not need to be connected. We add this restriction first, to ease the computation of a solution, and second, because what we want is not to reach a specific node but to have the sensor readings of all of them. Hence, we can just follow the same route forwards and backwards.

3. Restricted computation time. Albeit in our handheld devices a valid DFS solution appears in less than a second, with a little more time (∼20s) we can get, in some cases, a better result. We finally decided to fix a maximum waiting time of 30s, which gives enough time to compute a good solution in every case, and is short enough for a triaging member to wait for.

Likewise, we set the conditions to determine the fitness of an itinerary over another. In our case we want to minimize the amount of energy sensor nodes consume, thus increasing their lifetime. As all nodes are active and sensing victims' health, the only other variable we can improve to minimize energy consumption is the time spent in the transmissions. To do so, a good approach is to minimize the distance an agent has to move to reach the next node. With this condition we also reduce the errors made during agents' transmission, thus minimizing even more the time used for an agent to migrate.

3.3 Times and routes

For evaluation purposes we measured both genetic algorithm and DFS times. Our goal was to check the appropriateness of the solutions for our traveling agent in fully connected networks and in partially disrupted networks.

We started with the genetic algorithm in fully connected, randomly generated graphs. Tests started with initial populations of 10 and 50 randomly generated individuals. Generations had a mutation probability of 0.01 and a crossover probability of 0.7.

First tests with fully connected networks performed well both in computation time and in path cost (see Table 3.1 and Figures 3.3a,3.3b). In the table each cost value is the mean of 5 executions of the algorithm in our test handheld.

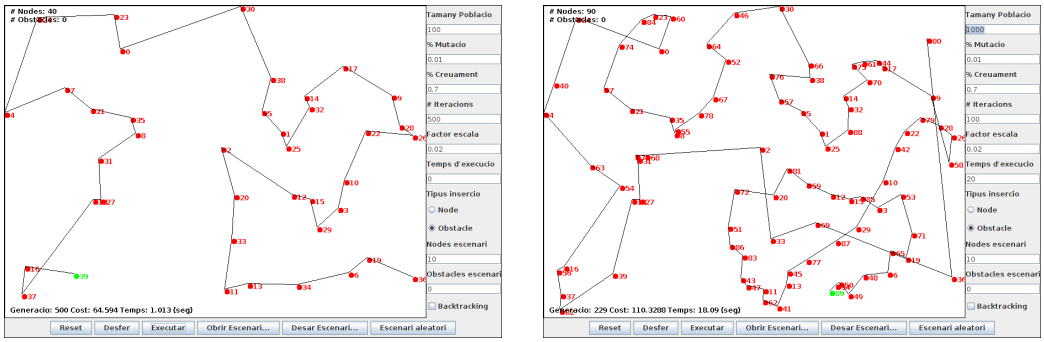
Fully connected graph - Genetic Algorithm - N810		
	10 individuals	**50 individuals**
5 Nodes	*cost*: 16.0755 *time*: 0.7540	*cost*: 15.5693 *time*: 6.0744
10 Nodes	*cost*: 31.5538 *time*: 2.7115	*cost*:31.3979 *time*: 13.0550
25 Nodes	*cost1*: 68.3264 *time1*: 13.2146 *cost2*: 55.7277 *time2*: 30.0000	*cost1*: 69.5351 *time1*: 15.0000 *cost2*: 59.3514 *time2*: 30.0000 *cost3*: 55.8480 *time3*: 52.9582
50 Nodes	*cost1*: 134.1062 *time1*: 15.0000 *cost2*: 106.7367 *time2*: 30.0000 *cost3*: 102.8928 *time3*: 42.7622	*cost1*: 115.9869 *time1*: 15.0000 *cost2*: 107.5450 *time2*: 30.0000 *cost3*: 101.5540 *time3*: 60.0000

Table 3.1: Results for the genetic algorithm in a fully connected graph.

Results show the best solution obtained for time limits of 15, 30 and 60 seconds. In some cases the best solution was found before reaching the limit, in other cases the algorithm is still computing better solutions after the timeout.

As we can see in Table 3.1, a solution is found in the first 15 seconds in a 50 nodes scenario, but waiting a bit more time, until reaching the 30 seconds, an, approximately, 20% improvement is easily accomplished while not trespassing the hypothetical time limit we established for the medical personnel to wait during a triage (30s).

We also tested the genetic algorithm in partially disrupted networks with the same configuration as in the fully connected tests, 10 and 50 randomly generated individuals, mutation probability of 0.01 and crossover probability of 0.7. In these cases, the genetic algorithm is unable to obtain a solution in a reasonable amount of time due to the excessive cost of validating every individual



(a) 25 nodes (b) 50 nodes

Figure 3.3: WSNs solved with the genetic algorithm.

and maintaining a constant number of members in the population.

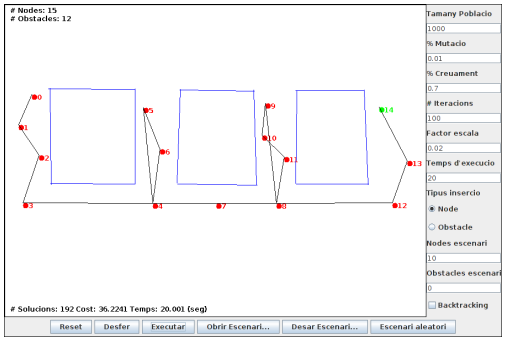
Nonetheless, DFS solved this situation and found good solutions in small amounts of time. Furthermore, in some cases, waiting for the time limit to expire, the solution improves considerably (see Table 3.2, where jNkO stands for j Nodes and k Obstacles).

Scenario with obstacles - DFS - N810			
5N1O Sc.	**10N1O Sc.**	**25N3O Sc.**	**50N3O Sc.**
cost: 21.995	*cost1*: 77.0664	*cost1*: 78.3977	*cost1*: 103.3174
time: 0.066	*time1*: 0.181	*time1*: 0.102	*time1*: 0.179
	cost2: 68.4044	*cost2*: 77.5867	*cost2*: 103.1823
	time2: 1.567	*time2*: 81.448	*time2*: 135.264
	cost3: 54.4441	*cost3*: 77.3539	
	time3: 5.351	*time3*: +300	
	cost4: 50.2357		
	time4: 203.782		
	cost5: 45.0835		
	time5: +300		

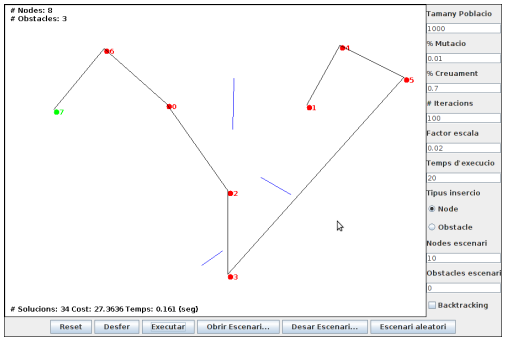
Table 3.2: Results using DFS in a partially connected graph

In randomly generated scenarios DFS proved its usefulness by generating useful routes for MAs to travel WSNs, both in fully-connected networks and in networks with broken links or visibility problems.

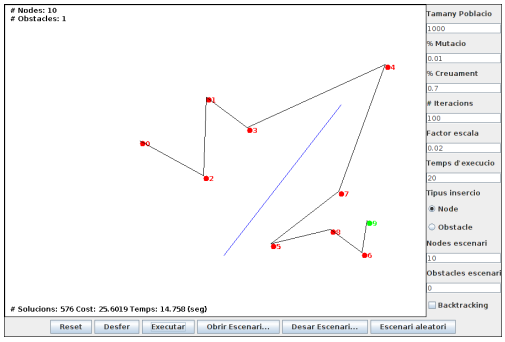
To further test the algorithm we generated special scenarios, easily found in actual catastrophes, Figure 3.4; reflecting a building, Figure 3.4a; a node distribution with one single solution, Figure 3.4b; two zones separated by a wall, connected only by one solitary node, Figure 3.4c; or a star shaped scenario, Figure 3.4d. DFS also found good enough solution in this type of scenarios in reasonable amounts of time. Table 3.3 summarizes its results.



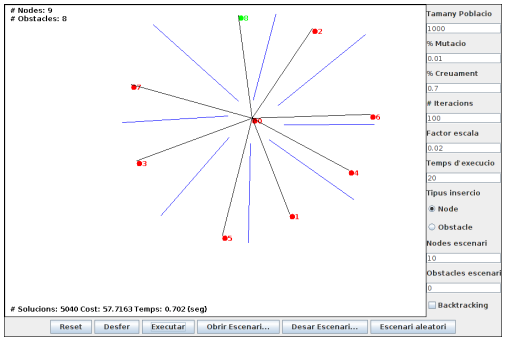
(a) Building-like.



(b) Unique Path.



(c) Wall.



(d) Star.

Figure 3.4: Special interest scenarios solved with DFS.

Special scenarios - DFS - N810			
Building	**Unique Path**	**Wall**	**Star**
cost1: 36.5687	*cost*: 27.3636	*cost1*: 27.6036	*cost*: 57.7163
time1: 0.08	*time*: 0.001	*time1*: 0.058	*time*: 0.061
cost2: 36.0348		*cost2*: 25.6019	
time2: +120		*time2*: 12.434	

Table 3.3: DFS results in special interest scenarios

3.4 Summary and Conclusions

In this chapter we presented an overview of a multiple agent heterogeneous architecture for victim monitoring in MCIs. We joined two different and unrelated agent technologies, JADE and Agilla, to create a fully functional system with real-time victim monitoring.

We have also seen that we can benefit of network knowledge in the handheld devices carried by triaging personnel to calculate an efficient route through the WSNs in very little time, that helps agents to effectively visit every node in the network without wasting time searching for unvisited neighbors.

Moreover, we also added fault tolerance methods to our node visiting algorithms so a failure on one or more of the nodes doesn't prevent the application from working and continue monitoring victims.

4

Mass Casualty Incident Management with Mobile Agents and Wireless Sensor Networks

THE first moments of a Mass Casualty Incident (MCI) aftermath are decisive to save the highest number of lives. As the resources at that moment are often scarce, it is essential to efficiently coordinate efforts and evacuate and give urgent treatment to the most severely injured, yet curable, victims. Therefore, the field work of trained personnel (doctors, nurses, paramedical, ...) triaging victims in accordance to their medical status is of capital importance. The information is later used to prepare a rescue plan for the victims to be moved to a safer zone. Traditionally, triage personnel used cardboard triage tags, Figure 4.1, to easily identify victims. These triage tags are filled following a standard triage method like Manchester Triage System (MTS) [MJMW06] or Simple Triage And Rapid Treatment (START) [Sup84], which result in a color code that is clearly shown in the cardboard tag, with some other basic medical information.

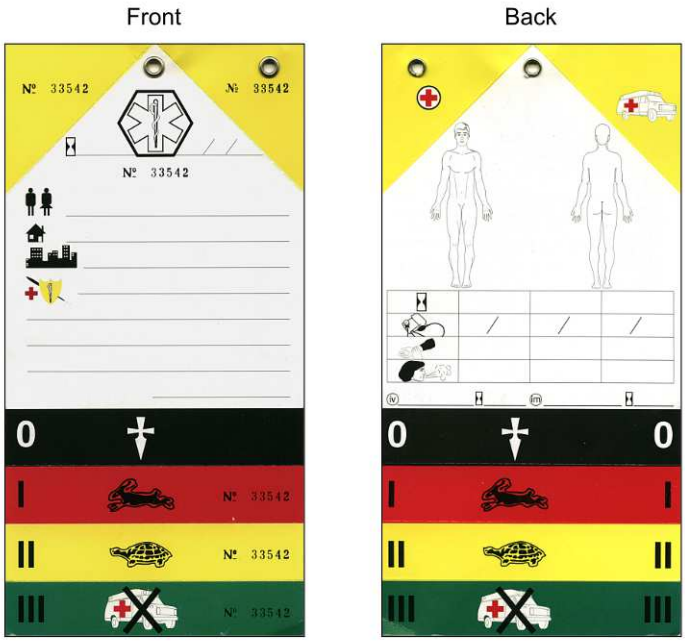


Figure 4.1: Front and back view of a traditional cardboard triage tag.

Information and Communication Technologies can greatly improve the efficiency of the work done by the trained personnel in these scenarios. Although, some tough limitations have to be taken into account. Trained personnel, for example, have to act quickly and will be reluctant to interact with complex system or fill in forms *in situ*. Regarding the technology, it should not rely on any local infrastructure, wired or not, as it could have been damaged in the casualty or be unusable, and setting up a new one can be very expensive in terms of money and/or time.

Mobile Agents (MAs) were previously introduced in this type of scenarios [MRMCC09], and showed that they can took great advantage of the technology. There, JADE mobile agents running in a touch screen handheld device, were used to bring victims' health state and GPS position to the Emergency Coordination Center (ECC) and, from there, plan an appropriate rescue plan. The traditional cardboard triage tag is still placed on the victim. JADE MAs makes its way to the ECC, leaping forward from device to device, lying stored in a one of them for a while, waiting for a proper candidate to jump to if it is at reach. The volatile network created with

handheld devices goes beyond *ad hoc* or Mobile Ad-hoc NETwork (MANET) possibilities, for the concurrent existence of communication links from source to destination is not required. Its routing protocol uses the time the device bearer will take to return to the ECC.

Albeit the proposal is a huge step ahead in MCI triaging it is still open to improvements. One of the main problems is that the carried information gets deprecated quickly, as victims' health status, which has a great impact on the subsequent rescue planning, may drastically change in a matter of minutes, and this changes are never carried to the ECC.

Just by adding health monitoring Wireless Sensor Network (WSN) nodes to the victims of the emergency we are able to update victims' health status stored in the ECC, and thus help the medical personnel to prepare a suitable rescue plan for the changing conditions.

4.1 Introduction

MAs are one of the most notorious examples of the flexibility of computer programming. The traditional paradigm of moving data to a computing center, with its associated costs, both of network bandwidth, time and storage, can be now turned over and move a lighter piece of operating code from data set to data set instead. By doing so, a lot of bandwidth and storage is saved, as well as possibly reducing the time by not having to wait for the data to be delivered, and by making calculations in a more distributed manner.

Itinerary planning is one of the key issues when designing WSNs MAs applications [CGL07]. Itinerary planning includes both the selection of the set of nodes to be visited, and the determination of the sequence in which they will be visited. Depending on how the itinerary is determined we can have *static*, *dynamic* or *hybrid* itineraries [CGL07]. In the *static* case, the decision relies completely on the *sink* node or base station **before** the agent is dispatched. If the itinerary is *dynamic*, the MAs autonomously decides the nodes to be visited and their sequence, according to the current status of the WSN. Finally, if we are talking about *hybrid* itineraries, we assume that the set of nodes to be visited is decided by the *sink*, but the sequence is determined dynamically by the MA.

Of the analyzed middlewares for MAs (see 2.1.3), only WISEMAN [GVVL06,

GVCL10] provides migration methodologies to support the three itinerary planning methods, *static*, *dynamic*, and *hybrid*. Nonetheless, in all the analyzed middlewares, the itinerary should still be defined inside the MA code.

In this chapter we adapt the separate itineraries firstly found in Concordia [WPW+97] (see Section 2.1.3) to a highly resource-constrained environment, i.e. WSNs, to improve the dynamism of the migration methodologies of WSNs applications. Moreover, we show how separate itineraries allow us to get a new application, exemplified with the Agilla middleware, in which agents move globally in a WSN while deciding the node-visiting sequence from these itineraries (*hybrid* planning).

4.2 Scenario

WSN describes the term of ubiquitous computing at it's best, small devices that can be easily situated in a myriad of places and program them to perform an innumerable amount of tasks. In this study, we propose a new method to share and aggregate data within WSN using MA itineraries and eliminating the need of a central collection, and failure, point: the *sink* node. This section describes the main actors and structures of an application of this kind.

Traveler MA moving through every node of the WSN using a previously calculated itinerary. It reads the data found in every node, and also writes the changes carried from previous nodes in the itinerary. Thus, every node of the WSN has an up-to-date log of the rest of the network.

Victim static agent residing in every node of the WSN reading sensor data and computing useful summaries with it. To save batteries it only retrieves sensor readings and computes the summary when the *Traveler* agent is in the same node, sleeping the rest of the time.

Extractor static agent residing in every node of the WSN waiting for a contact with an external entity to dump the aggregated contents when connected.

Delay and Disruption Tolerant Network (DTN) node portable device with WSN connection capabilities which fetches the data dumped by the *extractor*. It is also the responsible of computing the itinerary used by the *traveler* agent.

Itinerary data structure used by the *traveler* containing the list of to be visited nodes and its default visiting order. It is calculated by the DTN node when the WSN is fully set up.

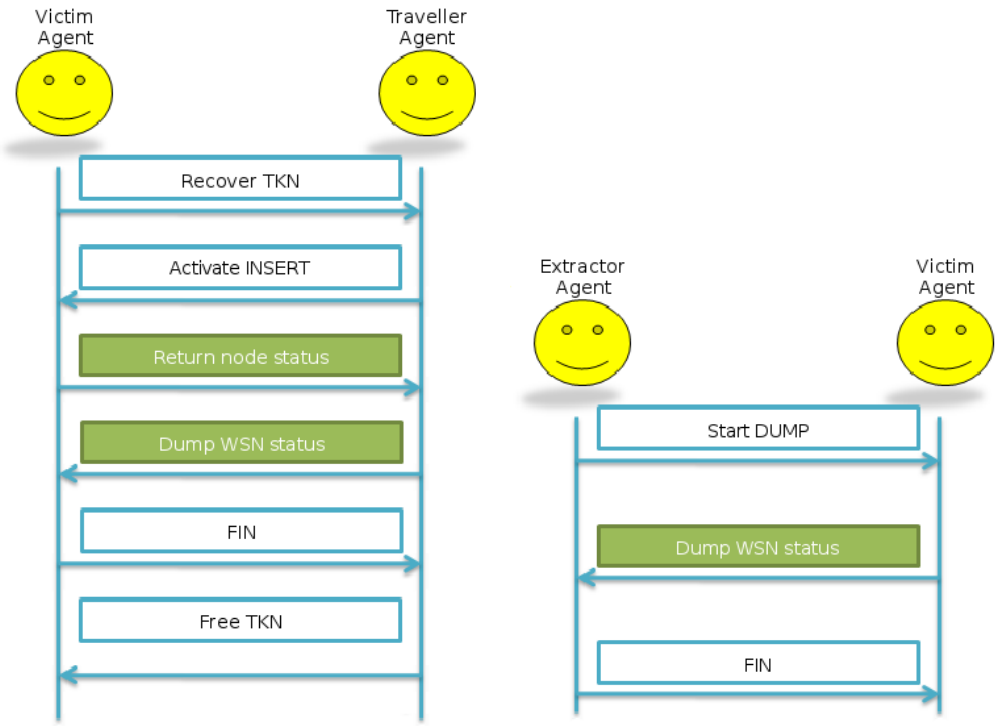
Agent communication is made through a series of steps depending on the actors and the data forming part of it. The communication between *traveler* and *victim* agents, where node readings are passed from *victims* to the *traveler* and the rest of the WSN status is passed from the *traveler* to *victims*, are *traveler* who initiate the protocol by sending a `Activate INSERT` message, *victim* then responds to this message with the sensor readings. When the last chunk of sensor reading data is received, the *traveler* starts sending the readings of the rest of the WSN nodes. Finally when the transmission ends, the *victim* responds with a `FIN` message, releasing the communication (See Figure 4.2a).

In its hand, the communication between the *extractor* and *victim* agents is simpler and requires just three steps. When in contact with a *victim* agent, found in every node of th WSN, the *extractor* sends a `Start DUMP` message, which the *victim* agent responds by sending back the status of the whole WSN. When all the data is received, the *extractor* agent sends a `FIN` message, which releases the communication. (See Figure 4.2b).

4.3 Separate itineraries for Mobile Agents in WSN

The small processing power and memory of the devices used in WSNs constrains the design of separate itineraries to fit in their limited resources. As our scenario requires active personnel carrying the monitoring nodes before its placing, we can assume that they may agree to carry, at most, the weight of 25 to 32 nodes (1.5Kg - 2Kg). With this limited number of nodes we can design an itinerary structure to use in memory constrained monitoring devices.

We managed to fit a single itinerary step into as little as 8 bits, including the node identification (5 bits), an on/off bit and information about the state of the monitored entity (2 bits). It is worth noting that the itinerary is not computed to be circular, thus, to continuously move through the



(a) Traveler - Victim (b) Extractor - Victim

Figure 4.2: Agent communication protocols.

WSN, the *traveler* agent has to traverse the itinerary forwards and backwards. Figure 4.3 depicts two positions of our itinerary structure using the aforementioned codification.

To adapt the itineraries to WSN nodes we made two simplifications to regarding those in conventional agents (see Section 2.1.3). Firstly, we considered only entries of the type *sequence* and *alternative*. *Set* entry types are out of the study as it will require cloning the agent a determined number of times and then gathering the collected results, as well as dealing with all its coordination, which will drain rapidly drain the battery of nodes. Thus, a WSN MA itinerary is a sequence of node identifiers stored in agents' memory. Agents will move to the next node in the itinerary and, in the case the next node is unreachable, or they decide otherwise, they will move to an alternative node.

The second simplification is to consider that the same code will be run on each node. Otherwise, nodes will require a capacity similar to Java reflection. However, it is easy to adapt agent's code to deal with specific nodes, e.g.

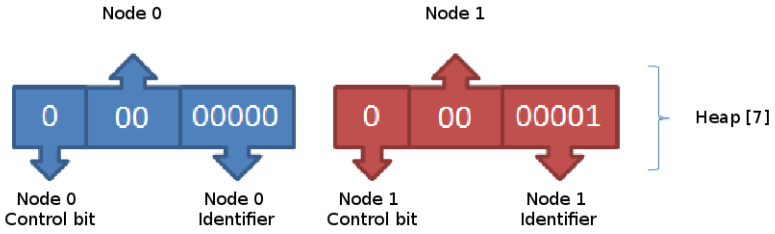


Figure 4.3: Sample itinerary entries using the proposed codification.

those at the ends of the itinerary.

4.3.1 Fault tolerance

Our proposal includes two fault tolerance mechanisms providing a MA is unable to follow the proposed itinerary, both oriented in preventing node failures or their unreachability. The first mechanism consists in moving to alternative entries in the itinerary structure, thus skipping failed or removed nodes. The second mechanism consists in using several *traveler* agents with different itineraries to minimize the probability of all agents being caught inside a failing node, and to cope with network partitioning.

In the first mechanism, when a *traveler* agent can not reach the next node in its itinerary, either because it is failing or because it was attached to a removed entity, it can apply up to three different methods to move to alternative nodes. In the first implemented method, **jump-after-next**, the MA looks a step forward into the itinerary structure and tries to migrate to the following node if available and in range. If it is not, the *traveler* agent decides, by a simple coin-flipping algorithm, to choose one of the remaining methods: **random-jump** or **reverse-itinerary**.

In **random-jump**, the *traveler* gets the list of available neighbor nodes and randomly chooses one of them for its migration. After the migration, the *traveler* agent tries to resume its sequential route from the new node. In the **reverse-itinerary** method, the trip through the itinerary is simply reversed, just as it is done when arriving to the end node. Note that this last method is a useful strategy when the network is partitioned, as both **jump-after-next** and **random-jump** always try to go forward from the new node, and thus

some backward nodes can be left unvisited.

For the second fault tolerance mechanism, our proposal can use several *traveler* MA, where each of them starts its itinerary from a different node of the network and, preferably, follows a completely different itinerary. This method adds another layer to the tolerance against failures of the application, increasing the robustness of the WSN in terms of partitioning. Now it is more unlikely for a WSN to end up with a unmonitored part due to a permanent failure of one, or several, critical nodes.

The alternative itineraries followed by the additional *traveler* agents are also calculated in the DTN node, which has all the information about the monitored entities. In this node, by selecting different starting nodes, we can obtain different, but similarly effective, itineraries for all the MA. To prevent DTN node bearers of having to move to several monitored entities to inject the additional *traveler* agents, we use the same strategy as in the **random-jump** fault tolerance method: each MA looks for the actual node in the itinerary and follows its route from there.

The drawback of using more than one *traveler* is that the integrity of the aggregated data is not guaranteed, i.e., an agent may update more recent data written by a previous *traveler* agent. Given the small number of nodes of our proposal, and that the roundtrip time for a MA in such small network is short this is not a critical issue. We could easily overcome this drawback by adding a **last-updated** timestamp into the data set.

A situation that we had to take into account is the meeting of two, or more, *traveler* agents into the same sensor node, if the node fails then all *travelers* inside it will be lost. This is specially critical if the failing node is the only link between two clouds of sensor nodes. To prevent this accumulation of MAs we use a token-like solution. When a *traveler* agent wants to migrate to a new node requests the token, if it is available, takes it and continues its normal execution. If, in the other hand, another *traveler* agent has already taken the tuple, the new agent tries to move to another node using one of the aforementioned alternative methods.

4.4 An implementation using Agilla Mobile Agents

Using the only publicly available MA middleware for WSNs in TelosB nodes, we implemented a proof-of-concept application and taken measures of both running time and energy consumption to prove that our proposal is not only feasible but useful in real-life scenarios.

We performed simulations using TOSSIM and TinyViz, and using actual nodes both, in a closed environment and in a could-be-real scenario.

With the programming resources provided by Agilla we designed an efficient communication protocol between the agents of our application using reactions. In the communication between the *victim* agent and the *traveler* agent, a tuple matching a reaction is placed by the latter into the formers' tuple space to notify its arrival to the node. The *victim* agent then starts dumping its sensor readings using the protocol as depicted in Figure 4.2a from Section 4.2. In the case of the communication between *victim* and *extractor* agents, a different tuple matching a different reaction is placed by the latter into the formers' tuple space. The *victim* agent then dumps its collected data into the *extractor's* tuple space using the protocol depicted in Figure 4.2b from Section 4.2.

The 8-bit per hop itinerary structure detailed in Section 4.3 is stored in the heap memory construction into the Agilla MA. That way we can store the maximum amount of nodes we defined for our architecture (25 to 32) in just 16 positions of the heap, and still have free positions for other application-related purposes. The random-access nature of the heap permits agents to move throughout the network in whichever way they need, a very useful property for the defined fault-tolerance mechanisms.

What is more, using 8 bits to define a step of the itinerary, we can fit two hops in just one position of the Agilla stack. Unfortunately, albeit this is an advantage when looking to memory efficiency, this complicates the reading of an itinerary step from the heap, forcing the division of the value and choosing the required bits after pushing it onto the stack. Luckily, the Agilla ISA offers a handy `shiftr` instruction which we use to move the value and get required bits.

4.4.1 Memory usage

In our implementation, the memory requirements of the application are very small, leaving room in the WSN for a broad range of improvements. A basic Agilla installation of our implementation in a TelosB node takes up to 3866 bytes out of the available 10KB of RAM, and 45308 bytes out of the available 48KB of ROM, increasing in just 400 bytes of RAM a clean Agilla installation.

When deploying the application we ran into some issues regarding limitations of the platform to host a large number of agents and neighbor nodes, regarding the size of the messages exchanged between agents, and regarding the size of the tuple space. To circumvent them we had to make some configuration changes, the most important being:

- `$AGILLA/nesc/agilla/Makefile.AGilla`
  - l. 1 `-DAGILLA_NUM_AGENTS` from 3 to 6
  - l. 2 `-DAGILLA_NUM_CODE_BLOCKS` from 12 to 60
  - l. 14 `-DAGILLA_MAX_NUM_NEIGHBORS` from 20 to 25

- `$AGILLA/nesc/agilla/types/TupleSpace.h`
  - l. 45 `AGILLA_MAX_TUPLE_SIZE` from 20 to 48

- `$TOSDIR/types/AM.h`
  - l. 65 `#define TOSH_DATA_LENGTH` from 29 to 36

With this small changes we increase the maximum number of agents per node, from 3 to 6, the maximum memory used by an agent's code, from 12 to 60 code blocks (1 code block equals 22 bytes), the maximum number of neighbors per node from 20 to 25, the maximum size of a tuple from 20 to 48 bytes, and the length of a TinyOS message from 29 to 36 bytes.

When agents are injected into the node, they are saved in node's RAM, occupying the memory needed by the code (26 blocks, or 572 bytes) from that reserved by the platform, plus the stack, the heap and agent's registers (248 bytes).

4.4.2 Experimental evaluation

We evaluated the performance of our implementation against the TinyOS simulator (TOSSIM), a testbed of up to 25 nodes, and in an open field scenario that closely resembles a MCI situation.

The goal of the evaluation was, apart from debugging and checking the correct working of every part of the application, to check the response of fault-tolerance mechanisms to node failures, and to measure round-trip times of our *traveler* agents in different WSN configurations.

We first simulated the application using TOSSIM and TinyViz, the simulator and visualization GUI for TinyOS. Debugging with this tools is not very straightforward due to the ill-defined error messages thrown by Agilla, being `INVALID_TYPE` and `INVALID_SENSOR` the most common, with no other information, not even the line number. This forced us to follow the code step by step and picture the stack and every memory position at every time.

When the code was finally checked and working we built a testbed of up to 25 TelosB nodes (Maxfor MTM-CM5000-MSP and MTM-CM4000-MSP) running Agilla v.3.1.1 over TinyOS, and injected our agents there to check their correct behavior on actual nodes, and to check that the results obtained with the simulations were valid on real sensor nodes. Unfortunately, this move forced us to make important changes to our design, such as moving from reactions to active waiting in *traveler* agents, due to problems during their transmission, and reducing the size of TinyOS messages.

After verifying the proper functioning of the applications with the changes applied, we moved our application to a real deployment where we tested our application in a 15 node scenario which closely resembles a part of a MCI.

4.4.3 Simulations

In the first simulation of our application using TOSSIM and TinyViz, we used a circular topology with 10, 20 and 25 nodes (Figure 4.4), all of them properly functioning. These helped us to solve small programming issues, problems with the stack and with conditional and non-conditional jumps, most of them due to the length of the code being left behind, easily solved using a longer jump instruction.

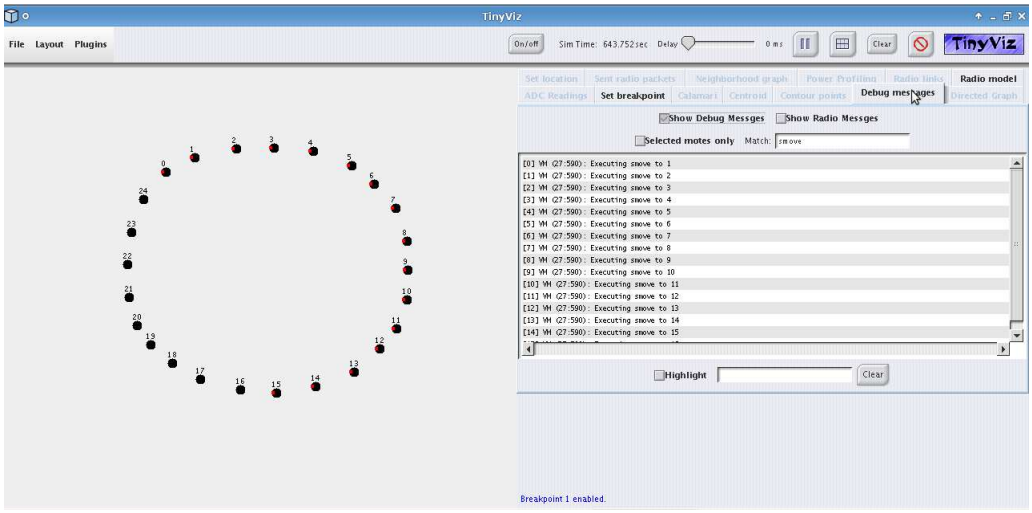


Figure 4.4: 25 node simulated circular topology.

After that we used the same circular topology but started forcing the failure of randomly selected nodes. The implemented fault tolerance methods behaved as expected, avoiding the problems of not finding the expected next node and having to migrate to a different one, either looking forward one position in the itinerary or randomly jumping to a reachable neighbor. This tests also proved the completeness of the fault tolerance methods when the *traveler* agent resumed its operation after an alternative migration.

The last simulation was done using two *traveler* agents in a partitioned WSN (Figure 4.5). We tested their correct adaptation to a failure of an important node, a node being the only link between two sections of the WSN. After the failure, both agents remained in its partition of the network, visiting their subset of nodes either following the itinerary either applying one of the fault tolerance methods when not possible.

These simulation proved that our application worked correctly on actual nodes and that was capable to respond to unforeseen situations. To calculate its performance in terms of time and energy consumption, though, we moved to a more appropriate close-to-reality scenario.

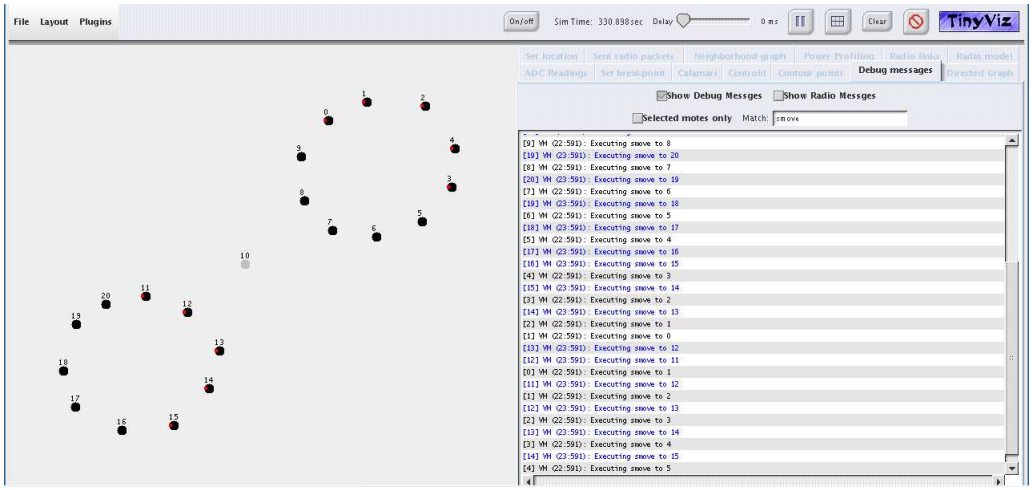


Figure 4.5: 21 node simulated breakable topology.

4.4.4 Testbed runs

The first test on the testbed consisted in verifying the application to work as in the simulations, without measuring time nor energy consumption. First we made some test runs with all the nodes working and fully functional. At that moment we perceived a problem with *traveler* agents and their reactions. The problem was that after running the application for an undetermined time, the reactions in the *traveler* agent stopped communicating with its bearer, resulting in a complete stall of the application. To solve the issue we moved communications done by these reactions to an active polling method, where the agent looks for the expected data in the tuple space. With the problem circumvented, we started measuring the time needed for *traveler* agents to visit each node of the WSN.

Results of the runs with 10, 15, 20 and 25 nodes in a WSN without failed nodes showed that a complete roundtrip of the *traveler* agent takes 1.2s per node in the case of a 10 node network (24.4s until returning to origin), 1.46s in a 15 node network (43.8s total), 1.45s in a 20 node network (58.1s total), and 1.59s in a 25 node network (79.3s total). The increase in the time spent per node is caused by the bigger amount of data carried by the agent, mainly larger itineraries and more sensor readings.

Table 4.1 depicts the running times of the aforementioned tests, the samples being the mean of 5 runs of 10 roundtrips each, error bars show the deviation of the data in each roundtrip sample.

# nodes	time (s)
10	24.419 ± 1.5s
20	56.786 ± 1.8s
25	1:20.275 ± 2s

Table 4.1: Testbed roundtrip times after 10 runs.

The second set of tests was done in a problematic WSN, that is, one with failed or malfunctioning nodes. First, we benchmarked the performance of our *traveler* agent with some deactivated nodes, 5 in the case of the 10 node WSN, 9 for the 20 nodes one, and 11 for the 25 nodes WSN. These runs were aimed at measuring the correct working of the **jump-after-next** fault tolerance method, thus never deactivating two consecutive nodes.

Results, depicted in Table 4.2, show that the *traveler* agent responds very well to problems with non-consecutive failing nodes, reducing roundtrip times by nearly the half when compared with the WSN without failed nodes. This reduction happens mostly because the larger amount of time is spent doing migrations thus, the less nodes to visit, the less migrations to perform, and the less time to complete the roundtrip.

# nodes	time (s)
10	13,541 ± 0.7
20	29.186 ± 0.9
25	39.356 ± 1.3

Table 4.2: Testbed roundtrip times with failed nodes.

Finally, the last test applied to the testbed was focused on measuring the response of the application to network partitioning. The deployment for this test consisted in two separated clouds of 10 nodes each, connected only by a single critical node. We injected two independent *traveler* MAs, with different itineraries and, after that, we turned off the critical node deliberately. Each *traveler* agent was left in a different network partition, being completely disconnected from the other part of the WSN. We proved that having two independent *traveler* agents with disjoint itineraries is a good solution for this kind of situations where the itinerary is not complete in any of the partitions and fault tolerance methods are heavily used. We also used this scenario to measure the time needed to visit every node of a partition, with the *traveler* agent being forced to use every fault tolerance method implemented.

In this case, we consider a roundtrip done when all the nodes of the partition cloud have been visited. Results showed very variable roundtrip times, due to the heavy use of the **random-jump** fault tolerance method, which doesn't assure that a non-visited node will be selected as the next destination (Table 4.3. Nonetheless, the test was useful to confirm that the application is robust enough to undergo situations of this kind.

Lap	Lap time	Total time
1	01:02.832	01:02.832
2	00:55.650	01:58.482
3	00:59.964	02:58.446
4	00:41.615	03:40.061
5	00:21.669	04:01.730

Table 4.3: Testbed roundtrip times with 21 disconnected nodes with one *Traveler* per cloud.

4.4.5 Real world scenario

Finally, we tested our application in a real world scenario, modeling a MCI outside the building of our college (Figure 4.6). For our tests we used 15 sensor nodes and tested the correct working of the application both, with all the nodes working, and deactivating some of them. We asked some colleagues to hold one of the sensor nodes and to act as a victim of a MCI (Figure 4.7 (Photo taken with a fisheye lens. Distances may appear distorted. GPS coordinates for an accurate view of the area: (41.499727, 2.112164), (41.500070, 2.113401)).

Tests carried out with all the nodes working proved that what was promising in the testbed is also applicable to a real scenario. Roundtrip times for the application in a fully active network, with any failing node, also show that testbed roundtrip values can be extrapolated to real scenarios. Runs of 10 continuous roundtrips produced times of around 39 seconds for the 15 nodes WSN, barely more than one second per node, a value very similar to that obtained in the testbed.

Next tests were aimed to measure the overhead produced by applying fault-tolerance methods to the itinerary roundtrip. Using the same building scenario, we conducted another test where we randomly failed 5 nodes, never

Figure 4.6: Google Maps capture of the scenario location.

consecutive. Results are shown in Figure 4.8, where three different graphs are depicted. One for the times of our application running on a 15 node WSN, another showing a 10 node WSN, and a third on showing a 15 to 10 node run, progressively failing nodes. The two first values correspond to the WSN without failed nodes, following a one node per run fail until reaching the final 10 node configuration. In the figure can be seen that using the **jump-after-next** fault tolerance methods does not increase significantly the time needed to make the roundtrip to the whole WSN, see last four samples. This is because the most time consuming task of our application are the transmissions required to migrate the *traveler* agent and its data.

Tests with consecutive failed nodes have been also performed confirming the usefulness of the application in these cases. Roundtrip times, as they were on the testbed simulations, are variable, and differ a lot from those seen in the other tests. They are not suitable for benchmarking purposes, but they prove the resilience of our application to all kinds of node configurations. Table 4.4 shows the roundtrip times obtained when traveling the broken network on the building scenario.

Figure 4.7: Field deployment.

Lap	Total time	Lap time (s)
1	2:26.246	2:26.246
2	4:30.442	2:04.196
3	5:47.146	1:16.704
4	6:20.464	0:33.318
5	8:02.186	1:41.722

Table 4.4: 15 node building like scenario with 5 consecutive failed nodes.

4.4.6 Energy consumption

In our tests we used TelosB compatible nodes powered by two AA batteries (3V), consuming roughly 29.2mA when receiving to 20.6 when transmitting, according to the manufacturer product reference guides. Experimentally, we reached a mean continuous operation of our application (i.e., a continuous migration of our *traveler* agents), in a scenario with 15 working nodes powered by standard AA alkaline batteries, of nearly 5 days. Similar lifetimes are praised by other existing WSN applications for emergency scenarios [GMS+07], and easily more time than the expected to rescue triaged victims in a MCI.

4.4.7 Deployment issues

In some cases, we observed an interesting behavior when injecting agents in a newly created WSN. When testing a network with failed nodes to check

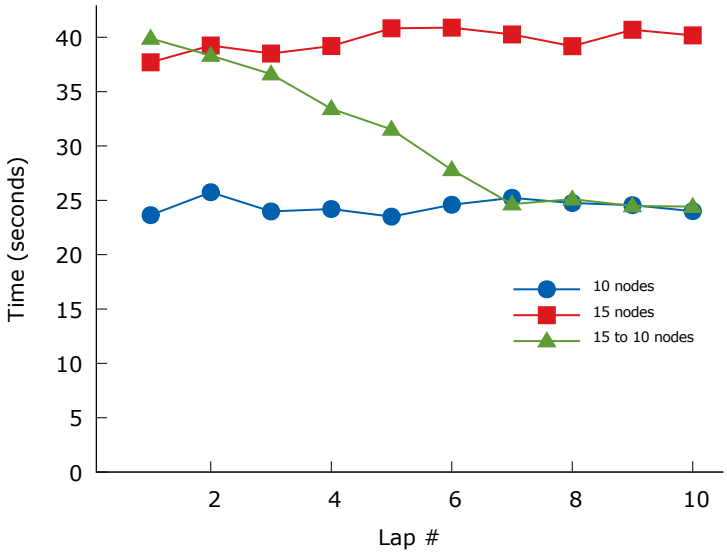


Figure 4.8: 15 node circular topology with failing nodes comparison.

the behavior of the **random-jump** fault tolerance method, the *traveler* MA got lost for a while, then reappearing following the pre-established itinerary as if nothing happened. This issue raised important headaches to the team, forcing the reprogramming of the itinerary, recalculate its values, etc. At long last, not before some discouraging tests, an actual light appeared in our testbed, not in a node from the studied WSN, but in a foreign node, not used in the current test but powered on. The *traveler* MA, in one of its **random-jumps**, reached a neighbor node out of its WSN, and jumped to it.

After that, we ran more tests disconnecting every unused node and everything worked as expected, finishing an, in the end, enriching experience which opened an interesting research topic.

4.5 Summary and Conclusions

In this chapter we have seen how the adaptation of conventional separate itineraries to the Agilla middleware is both feasible (in space and time), and useful.

Regarding feasibility, in a highly resource-constrained environment such as

Agilla running on TelosB nodes, we have developed an application in which we can store separate itineraries with a length from 2 to 512 positions. Moreover, we experimentally found that our *Traveler* MA (of 820 bytes) is able to follow this itinerary with an approximated time of migration between nodes of 1.5 seconds.

Regarding usefulness, the inclusion of the separate itineraries (with their sequential and alternative entries) in Dynamic MAETT has shown the utility of these itineraries as they allow the application to be very fault tolerant, reacting in front of failing WSN nodes. WSN partitions can also be easily handled by the application just with two MAs following different itineraries.

All fault-tolerant strategies have been tested and all worked properly, allowing to keep visiting all the nodes of all the partitions in a variable but limited time. This time depends on the necessary number of random jumps for the alternatives, and on the number of coincident nodes in the different itineraries of all the *Traveler* agents.

In addition, we found that the middleware Agilla is flexible and robust enough to support a new application following a new approach with agents moving through the whole WSN according to separate itineraries. These itineraries have been easily incorporated to Agilla, and this leads us to believe that they can also be included in other WSN mobile agent systems running on less restrictive environments.

One of the drawbacks of our application is the need to leave the injecting node, the one attached to the handheld device, in connection range of the created WSN. This forces the medical personnel to use that node as part of the WSN, having to replace the injecting node every time a new WSN is created. Albeit the injecting node ends up working as any other node and the medical personnel does not have to carry any extra weight for this matter, the need of changing it every time a new WSN has to be created adds an extra task to the medical personnel. Not leaving the injecting node in connection range makes the application to behave strangely after some time, to finally end up totally motionless. Without having more Agilla internal information regarding this issue, we figured out that the injecting node is acting as a kind of a necessary *cluster head* for the WSN.

Moreover, right now, we only have seen the application working with a single WSN. In next chapters we will see how we can modify the current architecture to several independent (or not) WSNs.

5

Clusters in WSNs using Mobile Agents

WHEN first responders start victim triage in a Mass Casualty Incident (MCI) they rely on their sight to locate victims. Many nearby victims may be left unattended by one of the first responders and be later triaged by another one, thus ending up belonging to different overlapping Wireless Sensor Networks (WSNs). The movement of Mobile Agents (MAs) between overlapping WSNs is desirable, but it cannot be done in an uncontrolled manner. Agents in a WSNs should not be allowed to randomly migrate to a different WSNs if no suitable node is found in theirs.

The itinerary structure presented in Chapter 4 is well suited to define intra-WSN routes, but fails when a foreign node enters the WSN's radio range. This problem is not admissible in a rescue operation and we cannot force first responders to avoid such deployments, first because it is easy to miss a particular victim when in haste, and second, because victim's location is unknown.

Manual node placing in our MCI triaging system puts us in an advantageous position to easily build a clustered WSN. What we just have to prevent is the unwanted migration of MAs of a WSN to another WSN while maintaining the

flexibility, good roundtrip times and energy consumption of the non clustered system. Identifying WSNs with a unique ID allows us to do so, but now we have to deal with larger data sets, affecting both agents' migration times and energy consumption.

5.1 Introduction

Dynamic itineraries for mobile agents in WSN are a very interesting and robust way of collecting, aggregating and sharing sensed data in this type of networks. They facilitate field personnel job by maintaining an updated copy of the status of the whole network, and even remove the need of a dedicated node in WSN without affecting energy consumption. All of this is done in a record time, even useful in MCI where human lives are in danger.

A key issue that remained unanswered, though, is what to do when nodes belonging to different WSNs make contact, or when we want to use more than 32 nodes. Up to now, a MA living in a particular WSN makes no distinction about the nature of the node, and if, unfortunately, a foreign node reaches that network, the MA can easily jump to that new nodes, leaving its home WSN with no clear returning schedule.

WSN clustering is a widespread solution to improve their performance, management and size. How to assign nodes to a cluster and which one will act as its *sink* (cluster head) are open problems with multiple solutions proposed. In our case, WSNs are clearly delimited at deploying time and we can omit the node selection step. Which node will be the cluster head is also a solved problem in our case, as every node can rely data to an external node when requested.

What remains undone, then, is the definition of a MA itinerary structure which supports clustering. In this chapter we transform our previous itinerary structure to one that supports WSN clusters. All of this is done without losing the application resilience to node failures, as every fault-tolerance method developed from the previous structure is still valid for the new one.

5.1.1 Scenario

The scenario detailed in Chapter 4 where we had a static agent monitoring the victim, and a MA moving around a pre-built WSN, following a pre-computed itinerary, introduced an information technologies assisted triaging system for MCIs. There, first responders built multiple networks of small sensor nodes which continuously monitor MCI victims and send their updated health status to an Emergency Coordination Center (ECC) when possible using an opportunistic network of handheld devices worn by first responders themselves.

The size of each of this networks is purposely limited to 32 nodes, a weight limit imposed by the willingness of first responders to carry heavy weights. It is a problem though, that when nodes from different networks make contact, a highly probable situation, MAs pertaining to one network may accidentally migrate to another, leaving its own network partly (or totally) unmonitored. This situations must be avoided, as it is not desirable to leave possible curable victims unmonitored. However, it is valuable to be able to migrate a MAs from one network to another on purpose, being it a reallocation of resources in case of network partitioning, or an information exchange.

The itinerary structure presented in the previous chapter is not flexible enough to contain the necessary changes to support multiple overlapping WSNs.

5.1.2 New separate itineraries

Following the itinerary step design introduced in Chapter 4, and considering the limitations of the sensor nodes, we expanded the size of an itinerary position by 8-bit (16-bit total) to insert the network identifier, now being as follows: one control bit, two bits for the status summary of the unit, 8 bits for the network identifier and the last five bits for the node identifier inside the network. Of course the final user may decide to use the 13 identifier bits in another manner. In our case, due to the limitations stated in Section 4.3, the aforementioned structure is the most suitable, leaving space for up to 256 networks of 32 nodes each, that is, we are able to monitor up to 8192 MCI victims. Figure 5.1 depicts the new itinerary structure using our particular bit arrangement.

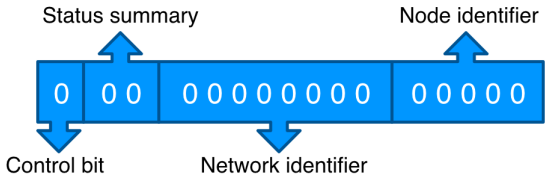


Figure 5.1: Generic itinerary entry using the new codification.

This new itineraries still use the same simplifications used in the previous structure, i.e., only entries of type *sequence* and *alternative* are considered, leaving *set* entry type out of the study due to its difficulty to work efficiently with battery powered devices; and that the same code will be run on each node, just adapting the code to deal with specific nodes, e.g. those at the ends of the itinerary.

As in the previous structure (Section 4.3), we also applied fault tolerance to cluster-aware itineraries. **Jump-after-next**, **random-jump**, **reverse-itinerary** were successfully ported to work with more complex structures. See Section 4.3.1 for a full description of these methods.

5.2 Implementation in Agilla

As we have done for the non-clustered construction, we simulated the new itinerary structure with TOSSIM and TinyViz, and in a closed environment. We used the same communication protocol as used in the non-clustered implementation (Figures 4.2a and 4.2b).

In this case, though, we have a 16-bit per hop itinerary structure which is stored in the heap memory construction. To be able to create a network of up to 32 nodes we had to increase the number of heap positions allocated by Agilla from 20 to 40; and the tuple space size from 100 to 170. Finally we have to modify the length of TinyOS messages from 29 to 60.

- $AGILLA/nesc/agilla/types/Agilla.h
  - l.138 `AGILLA_HEAP_SIZE` from 20 to 40

- $AGILLA/nesc/agilla/Makefile.Agilla
  - l.18 `-DAGILLA_TS_SIZE` from 100 to 170
- $TOSDIR/types/AM.h
  - l.65 `TOSH_DATA_LENGTH` from 29 to 60

After compilation of the application to reflect these changes it needs, in a TelosB node, 45356 bytes of ROM, slightly more than our non-clustered implementation; and 6170 bytes of RAM, around 2kB more than our non-clustered implementation. This increase is mainly due to the extension of the number of heap positions from 20 to 40, whose memory is pre-allocated when installing the application.

5.2.1 Experimental evaluation

We evaluated the performance of the clustered implementation in networks with up to 25 nodes for proper comparison with the non-clustered implementation, where we tested the application with a maximum of 25 nodes due to the supposed weight allowance of first responders.

Apart from debugging and checking the correct working of every part of the clustered implementation, fault-tolerance included, we want to measure round-trip times of the new *traveler* agent and compare them with the non-clustered implementation.

First, we simulated the clustered application using TOSSIM and TinyViz. After checking its correct working we tested the fault tolerance methods also in TOSSIM and TinyViz. Finally we moved our tests to actual nodes in a controlled testbed, where we took time readings for evaluation and comparison.

As in the simulations with the non clustered implementation we first tested the application with a circular topology of 10 nodes, all of them properly functioning (Figure 5.2). In this first simulation we used a simple itinerary sequence, where the *traveler* always jumps to its neighbor node in numeric order.

Confirmed the correct working of the implementation in a non-faulty network we moved our tests to a more complex one. In this case, we no longer used the simple circular network but a randomly generated one, disabled two of its

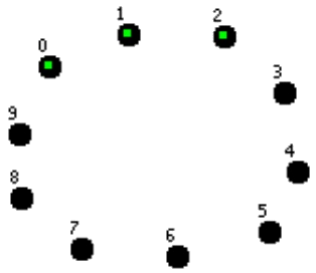
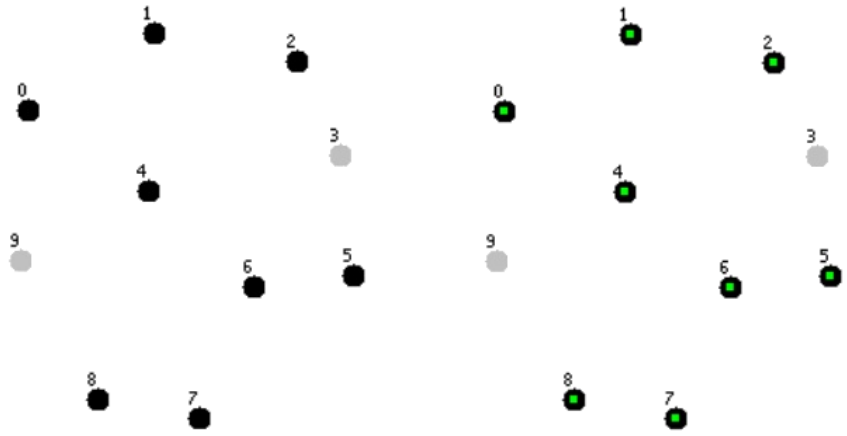


Figure 5.2: Circular topology with 10 working nodes.

nodes (Figure 5.3), and tested the **jump-after-next** fault tolerance method. As can be seen in Figure 5.3b, every node gets visited despite two of them being faulty, this was the expected behavior of the application and proves that a faulty node, if not critical, is not a hindrance for the MA to visit every other node in its itinerary.



(a) Nodes not visited. (b) All nodes visited.

Figure 5.3: Random 10 node network with 2 failed nodes.

The next test was designed to prove the correct working of the **random-jump** fault tolerance method. In this case we used a linear topology network and disabled some of its central nodes. The test wanted to prove that if no next or after-next node of the itinerary is available to jump, a randomly selected node, inside the domain of the WSN is selected, given it is in range. Figure 5.4 depicts the final state of the linear network with all its nodes visited after the agent making use of the **random-jump** fault tolerance method. As it is shown, every active node in the WSN is visited, even though there is no

continuity in the itinerary sequence. It is worth stating that the destination node can not be predicted due to the random nature of the jump, and an already visited node may be selected to continue the itinerary from that point. Its nature makes it suitable as a fallback fault tolerance method, to be used only in case any other method is able to find an appropriate next node.

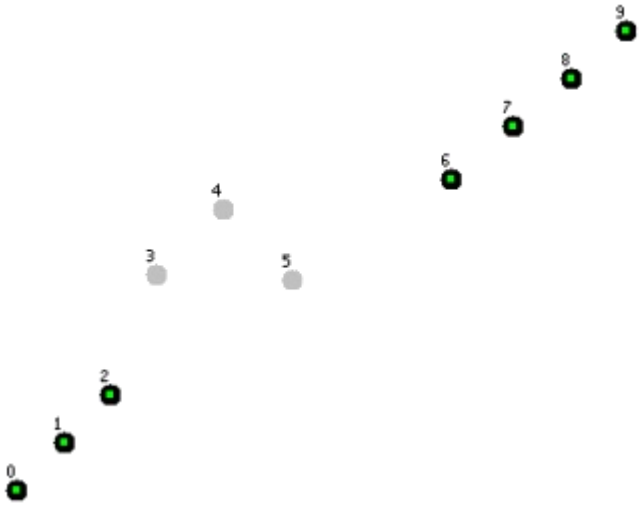


Figure 5.4: Linear 10 node network with 3 consecutive failed nodes.

Testbed evaluation

After checking the correct working of the clustered application in a variety of scenarios and with different node configurations, we compared the obtained results to our previous implementation. It is predictable that migration times, and subsequent roundtrip times, will be larger in the clustered approach, as the data carried by *traveler* agents is bigger. Remember that each itinerary step now takes a whole stack or heap position, while in the non-clustered approach they take just the half.

We ran the new application in 5, 10, 20 and 25 node WSNs and compared their roundtrip times to the non-clustered approach.

We expected to see a difference in migration and roundtrip times between the non-clustered and the clustered implementations increasing with the number of nodes. In the end, we are doubling the number of stack and/or heap positions needed to store the same number of itinerary steps in favor of

the flexibility of a cluster-ready WSN. Figures 5.5 to 5.8 effectively show the predicted differences. For example, for the 5 node network we see a mean increase of 3.22 seconds per roundtrip, averaging up to 11.8 seconds to perform the complete roundtrip. For the 10 node network the increase is even bigger, a mean of 8.82 seconds per roundtrip, now completing the full roundtrip in an average of 30.2 seconds. For the 20 node network we move to double figure numbers, reaching a mean difference of 17.77 seconds per roundtrip, finishing the roundtrip in around 77.1 seconds. Finally the difference for clustered and non-clustered application in a 25 node network is of the order of 24.74 seconds per roundtrip, completing it in 107.5 seconds in average (Table 5.1).

# nodes	Clustered time (s)	Non-clustered time (s)
5	11.8	8.58
10	30.2	21.38
20	77.1	59.33
25	107.5	82.76

Table 5.1: Clustered vs. non-clustered roundtrip times.

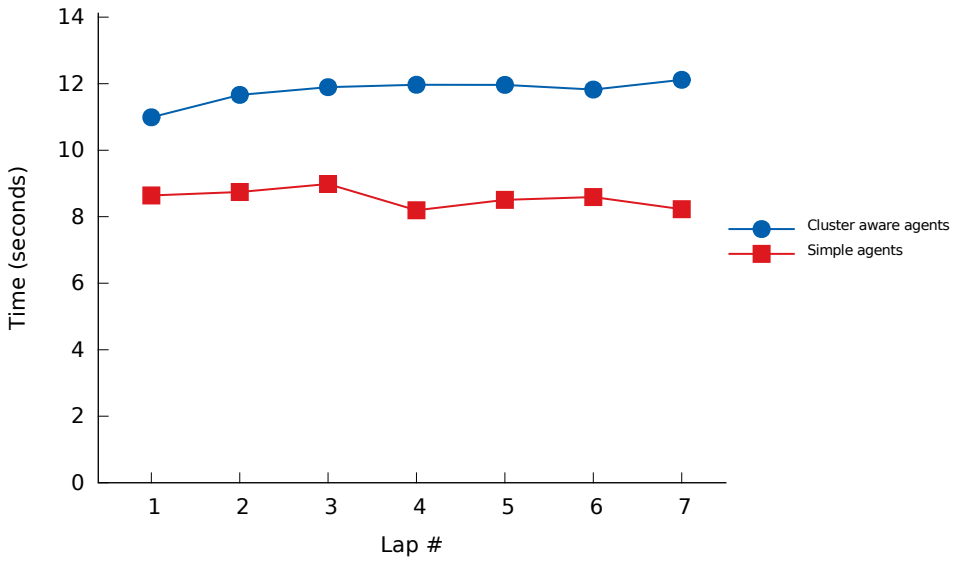


Figure 5.5: 5 node network clustered and non-clustered roundtrip times.

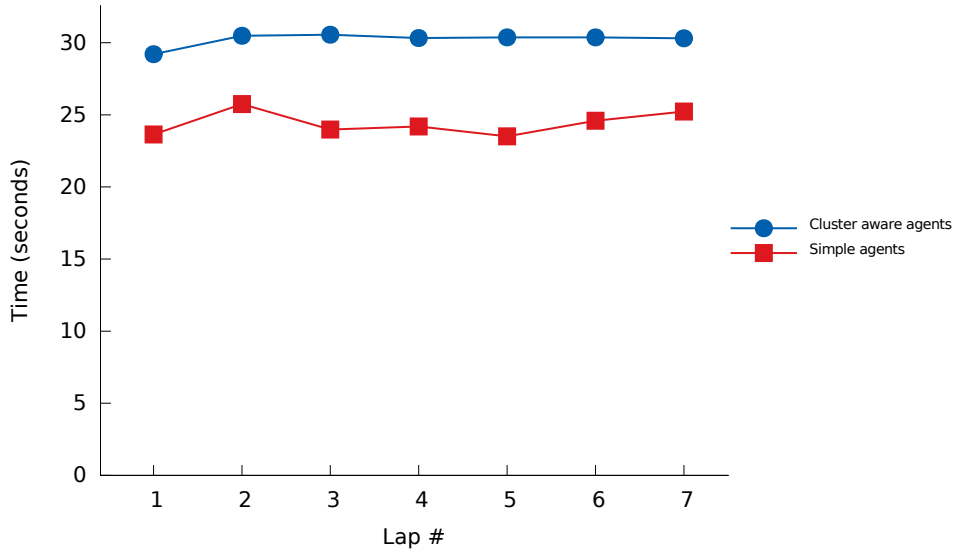


Figure 5.6: 10 node network clustered and non-clustered roundtrip times.

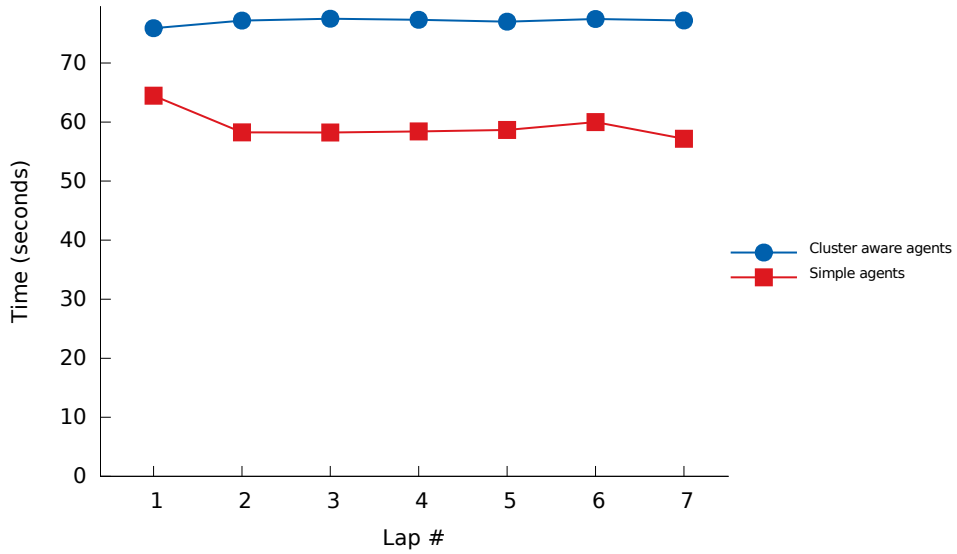


Figure 5.7: 20 node network clustered and non-clustered roundtrip times.

5.3 Summary and Conclusions

In this chapter we have extended the work done in the previous chapter to allow our architecture to accept more sensor nodes and overlapping networks.

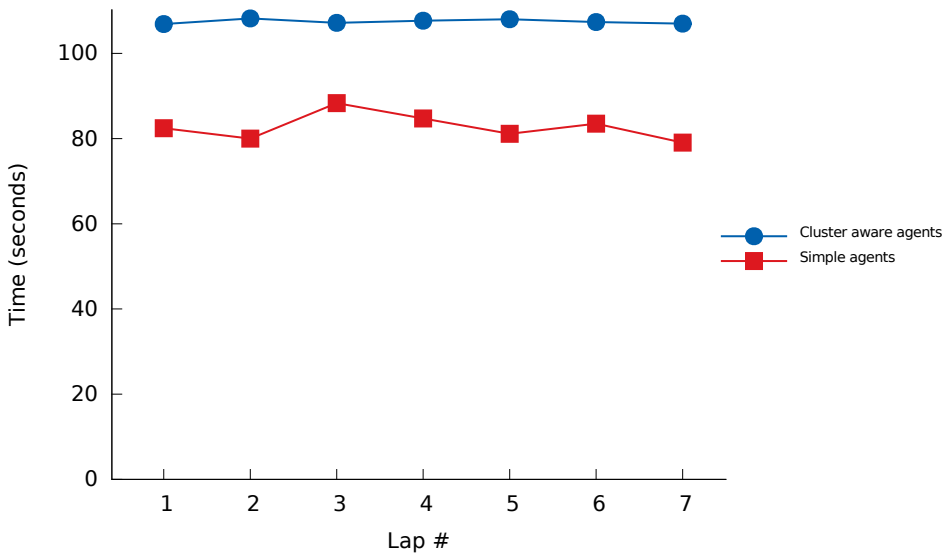


Figure 5.8: 25 node network clustered and non-clustered roundtrip times.

We proved that with little changes to our itinerary structure, we can expand our WSN to up to 8192 nodes. All this without increasing agent's roundtrip times significantly.

Concerning usefulness, separating nodes into clusters independently identified showed its utility in fault-tolerance tests, just affecting a single cluster, leaving the rest of the network unchanged, and by allowing more than the initial 32 monitored entities.

Moreover, we reproved that Agilla can support an application of this type, with complex itinerary structures to move independently through a large clustered WSN without ever colliding with agents from other clusters. And, being Agilla one of the lightest and resource restrictive mobile agent platforms, led us to think that porting these complex itineraries to other less restrictive mobile agent platforms will be a not-so-hard task.

6

Retrieval of Remote Moving Data in WSNs with Mobile Agents

NETWORKS where nodes don't have knowledge of the whole topology are, by definition, very limited in functionalities. Problems solved in traditional networks, such as routing, have to be rethought considering the particularities of this kind of networks. Finding a particular service in one of this networks is another, not tackled, problem. Imagine a node needs a particular service which is moving randomly through the network. It can of course, broadcast the request and flood the network with petitions, but it will rapidly saturate the network and block any other agent, or application, running on it.

Using some `clever` nodes in the network to contain more knowledge the location of about moving services makes contacting them easier and more efficient, while not considerably enlarging applications code. Once configured `forwarder` nodes will only make requests to clever nodes, which will respond with a route to the craved service.

6.1 Motivation

Fetching a distant service in Wireless Sensor Networks (WSNs) may be of great use in cases where services are limited and randomly moving through a large area, with no expected visiting time.

For example, in a Mass Casualty Incident (MCI), stabilizing victims before picking them up is paramount and any measure to increase the number of saved victims is very welcome. Right now, MCI victims are stabilized by qualified medics moving through the affected area and attending those victims they can find in eye range or using some Emergency Coordination Center (ECC) controlled assignment. In some cases, though, to properly stabilize a victim it may be needed a particular specialist (heart, lungs, ...) which may not be available in the surroundings at the time, thus leaving the victim not appropriately attended, or too late.

Using a service (medic) tracking algorithm in conjunction with a targeted message sending technique could highly improve the number of saved victims in scenarios of this type.

Current techniques are limited to the tracking part of the algorithm, not allowing to send any message to the target.

6.2 Scenario

In our MCI scenario we had paramedics going through the affected area stabilizing and triaging victims and equipping them with sensor nodes to monitor their vital signs, while they wait to be rescued. These paramedics, and therefore, victims, could take great advantage of this ability to call a qualified medic to properly treat an injured victim to be properly stabilized. Recall the scenario described in Chapter 3 and the triaging system described in Chapter 2, where every victim receives a sensor node which reads their vital signs and shares them with neighboring nodes, to rapidly deliver the whole WSN summary to a passing by paramedic equipped with a handheld device. Here we will use the nodes as a messaging infrastructure for medics and paramedics where paramedics will create the request and medics will receive it. For the infrastructure to work correctly we need to distinguish between two types of nodes, those who will act as simple senders or forwarders, and

those who will maintain an updated state of WSN's services: `forwarder` and `clever` nodes.

Clever nodes location is known by every node of the network, included those carried by the services, as well as a path to reach them. How clever nodes announcement is made to the rest of the network is not discussed here, as there exist a lot of alternatives in the literature, for example Directed Diffusion [IGE+03].

Clever nodes choice depends on the topology of the network, always looking for those nodes (one or more) that minimizes the overall number of hops to any service, a centroid search algorithm[KKRF12] for example. They, clever nodes, listen for services announcements and save their last known position, as well as the path to reach them, which is equal to the path the message from the service used to reach the clever node, and update entries accordingly. When they receive a service request, they respond the requester back positively if they can assign the service, or negatively otherwise, and forward the petition to the selected service, who then directs itself to the requesting node position.

Forwarder nodes, on its hand, just forward or make new service requests, sending them to one of the clever nodes and waiting for their response.

In Figure 6.1 we can see a sequence diagram depicting how a service is assigned to a forwarder node starting from its announcement. After the announcement to a forwarder node, service details are sent, through a multihop path, to one of the clever nodes in the WSN. There, it waits until it is updated by another announcement or until somebody requests their assistance. When requested the clever node sends back the assigned service ID to the requester and forwards the petition to the service.

6.2.1 Clever nodes

The poor knowledge WSN nodes have about the network they belong to difficult the correct routing of service requests and responses. That's why we introduce `clever` nodes, specially programmed nodes that wait for service requests and manage every aspect of the request, from service selection to notification to the requester.

Clever nodes listen for service requests from one of the other nodes of the

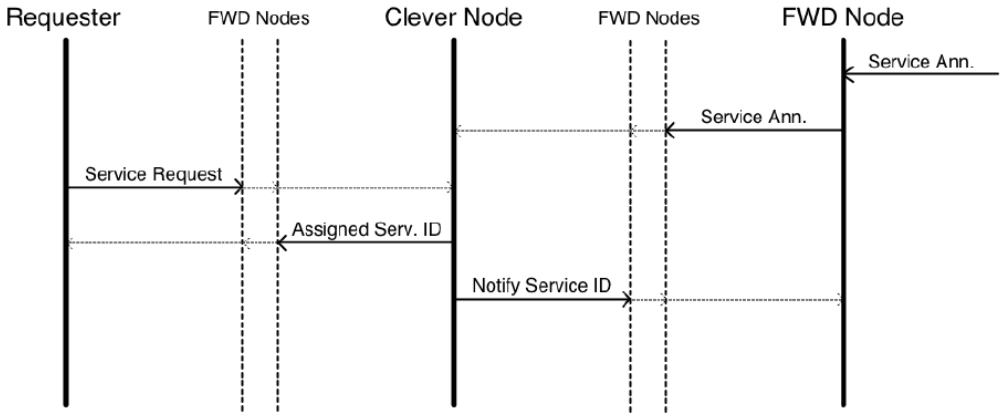
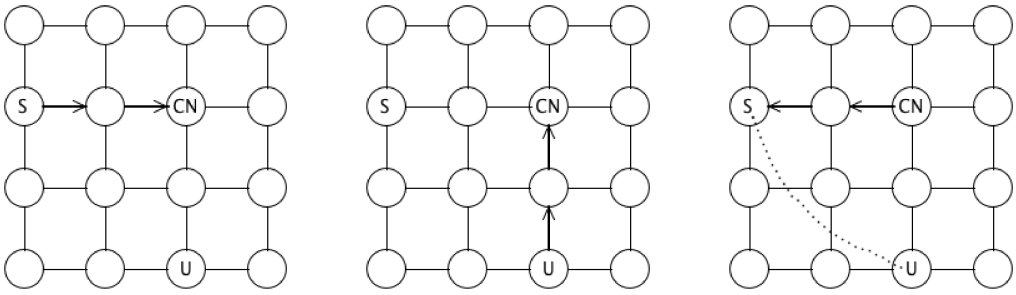


Figure 6.1: Clever node messages.

network and select an appropriate service from their list of available services depending on the requirements of the request and its location. Then they forward the received request to the selected service and notify the original requester that a service has been assigned. The assigned service is then removed from the list and further petitions for this type of service are assigned to another entry.

Services are added to clever nodes with simple announcements containing their identification, the offered services and the route they used to reach the clever node (Figure 6.2a).



(a) Announcement (b) Request (c) Selection

Figure 6.2: Service lifecycle.

Service selection needs to consider some important aspects, both of the origin of the petition and of the service itself, in order to minimize the time needed by the service to reach the requester. That is, a clever node will select the

service that minimizes the sum of the time to contact with it and the time needed for the service to move to where it is needed, or in other words, the proximity of the service to the request.

After assigning a service to a petition, or after a long time without contacting with a particular service, clever nodes remove it from its set of selectable nodes and, in the case of the service being assigned, start ignoring their update announcements. Also if clever nodes think that the number of hops needed to reach a service is excessive they will also remove it from its selectable set. Thus, clever nodes have three methods of removing a service from their selectable set, after it is assigned to a petition, after a time up since the last contact and if the number of hops is higher than a threshold. Figure 6.3 shows the lifecycle of a saved service.

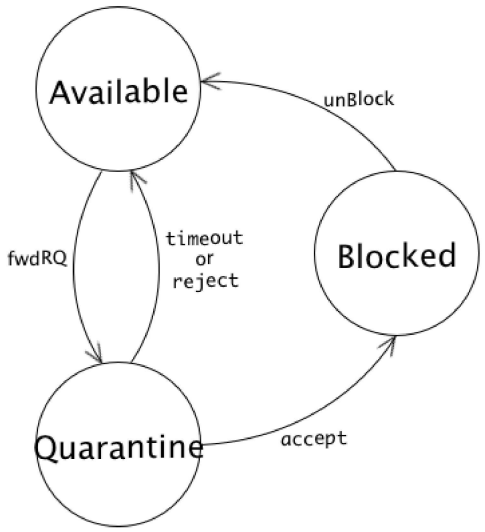


Figure 6.3: State diagram of a service in the clever nodes.

Entries in the selectable set are updated according to services' subsequent contacts, maintaining their identifier and offered service but resetting their timer and updating their route.

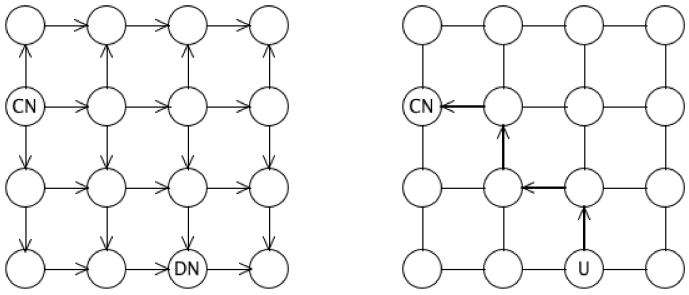
6.2.2 Forwarder nodes

While clever nodes wait for announcements and requests, and make decisions to efficiently manage them, the task of the other active component of the

architecture is limited to listen and forward announcements or requests to the next node in the petition's route to a clever node or to a service.

To forward requests to one of the clever nodes, they rely on a pre-computed next node which moves the petition closer to its destination. In our case we set up the routes using the first two steps of the directed diffusion algorithm [IGE+03]: interest propagation and gradients setup, with clever nodes as interest sources and forwarder nodes setting up their own gradients and choosing one of the shortest routes (Figure 6.4).

On the other hand, to forward a service assignment from a clever node to the last known position of the service, forwarder nodes use the saved route of the service announcement in the clever node and trace it backwards.



(a) Interest propagation (b) Gradients setup

Figure 6.4: Route definition from forwarder nodes to clever nodes.

6.3 Implementation in Agilla

To implement the proposed service tracking and retrieval protocol we used Mobile Agents (MAs) over the Agilla platform as in our previous implementations. In this case we programmed three different types of agents:

Clever node static agent In this case agents are static, they reside in the designated clever nodes listening for service announcements and requests. When a request arrives they assign a matching service.

Itinerary cloning agent Agent injected into the selected coordination points. They clone themselves to in range forwarder nodes and create a minimal hop route from forwarder nodes to one of the clever nodes.

Medic mobile agent Act as medics moving throughout the damaged area. If during their route they detect a node with special attention, they send a service request to one of the clever nodes.

Services mobile agent Act as medics offering services moving throughout the damaged area. A service announcement is sent every time they move from one node to another. They also listen for requests from clever nodes.

For the simulations we used, as in previous works, TOSSIM and TinyViz simulators. We started with a 25 node grid-like WSN with a single manually selected clever node situated in the center of the network.

In the clever node, for service selection we used a simple FIFO queue, but any more complex selection algorithm could be used, and every petition gets answered if a matching service is available.

Simulations started with the clever node announcing itself to the network, cloning a configuration agent to its one-hop neighbors, which keep cloning until every node of the network has stored the next jump, always keeping the one that minimizes the route to the clever node. When an agent arrives to a node with less or the same jumps to the clever node, it kills itself, not cloning anymore.

After that, we had the network set up. We then injected a single service to the network which started moving randomly through the network using its one-hop neighbors. After each jump, the service sent an announcement using Agilla tuples and reactions to the clever node using the route saved in the previous step which, when reaching the clever node, was saved into an Agilla tuple containing the service ID, last known position and the offered services.

Finally, we injected a medic agent which, as the service, randomly moves throughout the WSN using only its one-hop neighbors. At random positions it sent a service petition to the clever node using also Agilla tuples and reactions and waited there for the confirmation.

6.4 Evaluation

We didn't have any problem when simulating our approach with a small WSN (<= 32 nodes). Times were pretty good, reaching the sought service in times of the order of 1 digit seconds, which proved that our architecture and protocol were accurate to tackle this problem.

When moving to larger networks, though, Agilla began to complain about buffer and memory problems in the configuration part of the simulation. After some research we find that they were mainly because of the impossibility of the Agilla simulator to inject agents to nodes different than the gateway (node 0) without passing through it. Moreover, if nodes are not in connection range with node 0, even if there exists a path to it, the platform is unable to inject any agent to the remote node.

We modified Agilla's parameters to increase buffer and allocated memory, but it was all in vain, even though we managed to increase the number of simulated nodes up to 50 the platform kept complaining about buffer and memory shortage.

On the other hand, while testing for a way to inject agents one by one to remote nodes, increasing times between injections, increasing buffers, message and memory sizes, ... The simulator often crashed unexpectedly, forcing us to start the test from the very beginning.

We believe that it won't happen on physical nodes, as we can individually inject agents into their destination node, not needing to overload node 0. On the same matter, though, it may later raise problems with clever nodes not having enough memory to maintain a reasonable working set of available services. The unavailability of enough physical nodes prevented us to test these situations.

6.5 Summary and Conclusions

In this chapter we have implemented a working service retrieval system with mobile agents working on low powered sensor nodes.

We have seen that just adding a special logic to a small number of strategically placed nodes, we can easily find remote moving services in a network of indeterminate size. Unfortunately, in our tests we were not able to make it

work with more than 50 nodes due to simulator problems with the injection of multiple agents, though in the less than 50 nodes tests results were promising.

7

Access Control in WSNs using Mobile Agents

PROTECTING mobile agents has been, since their apparition, a widely studied topic [Vig97, ST98], even the protection of their itineraries [MB03]. Wireless Sensor Networks (WSNs) are not at all an exception, as they operate in open, usually non guarded environments and access to their information is easily accessible and altered. Traditional itinerary protecting schemes are hardly useful in these power constrained environments.

Public key authorization credentials provide a flexible approach to implementing access control in open distributed systems. WSNs being an example of such systems; however, their low-power nature implies energy efficiency requirements that may mean it is not practical to carry out computationally intensive operations, such as public key operations. In this chapter we describe a distributed access control system for WSNs applications that uses computationally efficient one-way hash-functions to implement authorization credentials in a highly resource constrained environment such as the one we use in this thesis by adding automation to victim triaging (see Chapter 4).

7.1 Motivation

Data and network access security in low power WSNs is, on occasion, something to take into account at the time of their deployment. The low computing power of their constituent nodes make common protection techniques to be revised and, more often than not, discarded, before applying them.

Despite the difficulties, it is possible, as well as desirable, to protect deployed WSNs information against unauthorized accesses, both for reading and for writing.

In previous chapters, we presented a new use case for Mobile Agent (MA) WSNs where sensor nodes conforming a WSNs monitor, record and transmit victim vital signs in Mass Casualty Incidents (MCIs). While nodes are used to process sensitive health information, privacy and security issues of the WSN are not addressed. Besides authentication and end-to-end security requirements, there is a need to provide support for authorization. In particular, a sensor node has to be able to determine whether it is safe to carry out some action on its sensitive data on behalf of some requester. For example, to permit a doctor's handheld device to modify victim's data on the sensor, or to integrate data coming from another sensor, or to accept a request to reprogram a hosted agent.

Existing authorization approaches are not practical for this scenario. While they may be interesting for traditional scenarios with non constrained devices, the nature of WSNs makes it impossible to apply them. Albeit it may be straightforward to implement static access control policies on a sensor node, these policies require, in practice, ongoing administration to reflect changing access requirements and direct sensor policy update or coordination with a central authorization server is not practical given the nature of WSNs.

In this chapter, we propose a Trust Management / decentralized authorization [BFL96, LABW92, EFL+99] style approach using public key authorization certificates to specify authorization delegation between public keys. As sensor nodes use low power processors with energy efficiency requirements, it is not feasible to carry out computationally intensive operations, such as those used in public key cryptography [Sen10].

We argue that Bloom Filters [Blo70] can be used to implement an effective access control system for this kind of applications which doesn't depend on

public key cryptography. Bloom filters are implemented using one-way cryptographic hash functions which are relatively cheap in terms of computation requirements, and are appropriate for WSN applications [Sen10].

7.2 Scenario

In the scenario detailed in previous chapters, we had a static agent in each victim monitoring them, and a MA moving around a pre-built WSN following a pre-computed itinerary. There, we introduced information technologies to assist in the triage system by using first responders to build multiple networks of small sensor nodes which continuously monitor MCI victims, and send their updated health status to the area's Emergency Coordination Center (ECC) using an opportunistic network of handheld devices worn by first responders themselves.

The access control method presented in this chapter uses the existing elements and infrastructure of the scenario to limit user's access to nodes, limiting their reading or writing permissions.

We use the ECC as the coordinating authority, having administrative rights for the entire system that decides the security policy/permission ordering. The ECC then delegates authority for sensor node access to their controlled nodes by administrative groupings. These groupings can then delegate authority to their teams/departments involved in the MCI which can, in turn, further delegate to other teams or individuals. Figure 7.1 shows an example for the used MCI scenario.

Sensor nodes then verify if a requester has permission to read or write their data.

7.3 Bloom filters

Bloom filters [Blo70] are space efficient probabilistic data structures used to test whether an element is or not in a set. As elements can only be added to the set, the probability of false positives increases with the number of added elements. False negatives, on its part, are not possible. A query to the set returns either *inside* (may be wrong), or *absolutely not inside*.

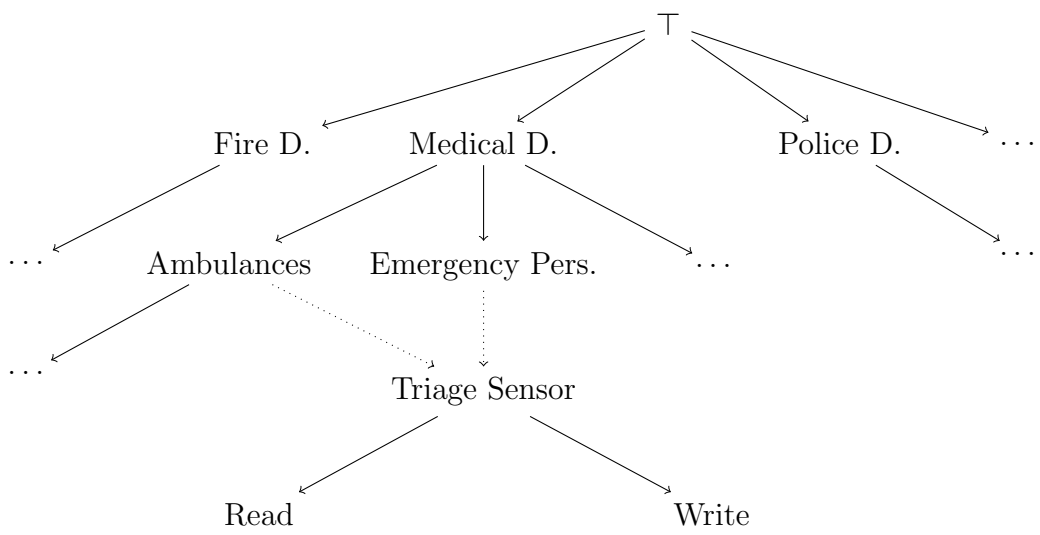


Figure 7.1: Permission scheme example.

A Bloom filter, when empty, is a bit array of a defined length m, with all its bits set to 0. To change the values of these bits we must define a number k of hash functions whose solutions map into one of the m array positions with a uniform random distribution.

Adding and querying elements of the set we have to feed it to every one of the defined hash functions to get k array positions, which will be set to 1 if we are adding an element, or compare with the values in the array to check if the element is in the set. If the comparison returns that the element is not in the set, we are sure that it is not, if, on the other case, the comparison returns that the element is in the set, it can be either because it is really in the set or, due to the addition of other elements, a false positive.

The example in Figure 7.2 we have a 18 bit Bloom filter array where we have defined 3 different hash functions. We have 4 elements (x, y, z, w) which we want to check if they are in the filter. We feed them to the hash functions and obtain 3 different array positions for each one of the elements. We then check if these positions are set to 1 in the array. Elements x, y, z have all their positions set to one thus, the filter returns true for these elements. Array positions for element w, however, have one of their positions set to 0 thus, element w doesn't pertain to the filter, and returns false.

Bloom filters have some useful security characteristics. Given a value it is fast and easy to check whether the element is in the filter. However, given

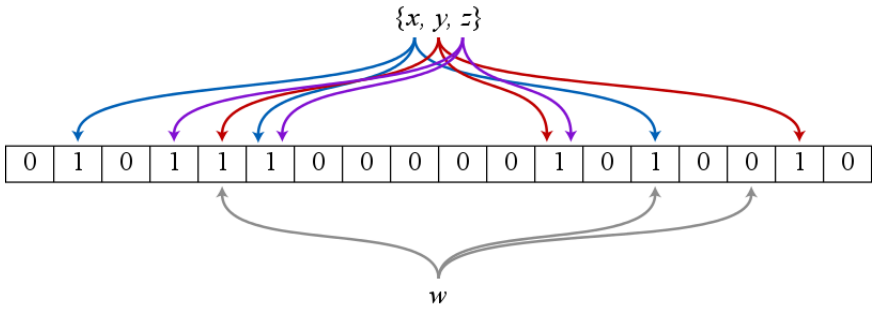


Figure 7.2: Example of a Bloom filter with matching and non-matching elements.

a suitably configured filter, it is not feasible to calculate the list of elements contained in the filter.

The idea of coding permissions in Bloom filters comes from micropayment systems [RS97, Ped97], where hash chains are used to represent coins, and can be used to provide a rudimentary form of authorization delegation [Fol03].

7.4 Bloom Permissions

We consider $(Perm, \leq)$ as a lattice of permissions P with a supremum denoted as $\top$ and an infimum denoted as $\bot$. Given $x, y \in Perm$ then $x \leq y$ is interpreted to mean "*if a user has permission y then implicitly he also has permission x*". Given a permission $a \in Perm$ then define $\lceil a \rceil = \{x \in Perm \mid a \leq x\}$ and we have that for $a, b \in Perm$ then $a \leq b \Leftrightarrow \lceil a \rceil \supseteq \lceil b \rceil$.

We also consider $\mathcal{P}(X)$ as a Bloom permission filter that is configured with the subset of permissions $X \supseteq Perm$, and $(\mathcal{P}, \supseteq)$ to denote the lattice set of all Bloom permissions. It follows that $a \leq b \Leftrightarrow \mathcal{P}(\lceil a \rceil) \supseteq \mathcal{P}(\lceil b \rceil)$. Thus, the set of all Bloom permissions $\mathcal{P}, \supseteq)$ is isomorphic to the set of all permissions $(Perm, \leq)$.

7.4.1 Bloom permission delegation

The administrative authority of the WSN and thus, the owner of the policy $(Perm, \leq)$, generates a set of Bloom permissions $\mathcal{P}(\lceil a \rceil)$ for each $a \in Perm$. The supremum permission $\top \in Perm$ is assumed to be a secret 'seed' that is

known only to the administrative authority. All other permissions in $P \setminus \{\top\}$ are considered to be public. Bloom permissions are intended to provide secret credentials that participants present in order to prove authorization for its corresponding permission.

Permission delegation Suppose a participant holding the Bloom permission $\mathcal{P}(\lceil a \rceil)$ wishes to delegate the Bloom permission corresponding to $x \in Perm$, where $x \leq a$ to another participant. The delegator does not know the secret seed $\top$, and is therefore unable to directly compute $\mathcal{P}(\lceil a \rceil)$. However, the delegator can compute the Bloom filter $\mathcal{P}(\lceil a \rceil) \cup \mathcal{P}(\lceil x \rceil \setminus \{\top\})$, which is equal to $\mathcal{P}(\lceil a \rceil)$, and pass it on to the recipient.

Authorization verification Suppose a participant presents a bit vector X corresponding to the Bloom permission $\mathcal{P}(\lceil a \rceil)$ as proof of authorization for an operation that requires permission $x \in Perm$. The recipient of the request checks $\mathcal{P}(\lceil x \rceil) = X$. If the recipient holds another permission $a \in P$ such that $b \leq a$, $\mathcal{P}(\lceil b \rceil)$ is easily computable from $\mathcal{P}(\lceil a \rceil)$, as $\mathcal{P}(\lceil b \rceil) = \mathcal{P}(\lceil a \rceil) \sqcup (\mathcal{P}(\lceil b \rceil) \setminus \mathcal{P}(\{\top\}))$.

Bloom permissions are intended to provide unforgeable secret credentials used to grant access to restricted resources. A participant holding the Bloom permission $\mathcal{P}(\lceil a \rceil)$ for $a \in Perm$ can compute any lower Bloom permission $x \leq a$. However, without knowing the secret seed permission $\top \in Perm$, or its corresponding $\mathcal{P}\lceil \top \rceil$, a participant holding $\mathcal{P}(\lceil a \rceil)$ cannot feasibly compute / forge the Bloom permission $\mathcal{P}(\lceil x \rceil)$ where $x \not\leq a$.

7.5 Access control in sensor nodes

In our MCI scenario we want to restrict the access to sensor nodes to be sure that data read and collected by them arrives safe and unmodified to the ECC.

We assume that, as the coordinating authority, the ECC has administrative authority for the entire system and decides the security / permission ordering. The ECC delegates authority for sensor access to the sensors controlled by administrative groupings, including the police department, medical services and fire department. These groupings delegate authority to the teams / departments involved in the MCI, who can in turn delegate to further teams

and individuals.

We also assume that sensors offer a set of actions to requesters (other sensors and readers) at their interfaces. If we let $(SPerm, \leq)$ define the ordering over corresponding sensor access permissions, for example, by a powerset lattice over its action interface, including, `rd` to read sensor data, `wr` to modify sensor data, `fwd` to accept data from a sensor and forward it, `install` to install a new mobile agent and `status` to read system status information. We assume there exists a secret seed action permission $\top_a$ that provides a unique greater bound on $SPerm$. Sensors are grouped according to its category. Let $(Cats, \leq)$ define an ordering between sensor nodes, and we assume there exists a secret seed category $\top_c$ that provides a unique greater bound on $Cats$. In principle, the kind of category chosen depends on the level of access control granularity required in the application, and can range from one single group / category, to a separate category for each sensor. For our scenario, categories are organized as a powerset lattice of MCI departments including `police`, `fire`, `medic`, etc.

The permission ordering for sensors is defined as a cartesian product ordering $(Perm, \leq)$ of the category ordering $(Cats, \leq)$ and the action permission ordering $(SPerm, \leq)$. A sensor in category c and offering an action that requires action permission p requires a requester to prove that it holds a permission (c', p'), or more specifically, the Bloom permission $\mathcal{P}(\lceil (c', p') \rceil)$, where $(c, p) \leq (c', p')$. For example, a radiography sensor checks that a requester holds permission $(\{\texttt{medic}\}, \{\texttt{rd}\})$ before responding with a heartrate reading.

During the initial configuration of a sensor that has to be assigned to category c, the ECC pre-computes a table that maps each action (permission) offered by the device to its corresponding Bloom permission for its category. For example, a mapping from `rd` action to $\mathcal{P}(\lceil (\{\texttt{medic}\}, \{\top_a\}) \rceil)$ to an authority in the `medic` department, authorizing it to initialize its own sensors.

Personnel authorization for some action on sensors is done by granting the corresponding Bloom permission. Here we assume that each participant holds a permission (c, a) and therefore, can request any sensor action requiring permission $(c', a') \leq (c, a)$. For example, a medic holding a Bloom permission $(\{\texttt{ambulance}, \texttt{police}\}, \{\texttt{rd}\})$ can read sensors placed by police and ambulance departments. If a participant holding permission (c, a) is delegated the new permission (c', a'), then they hold the permission $(c, a) \sqcup (c', a')$, whose corresponding Bloom permission is easily computed by a set-union. Further

research will include a role-based policy mechanism whereby participant's roles decide the permission that the participant holds.

The access control mechanism also applies to requests between sensors. In addition to mediating access requests (with their corresponding Bloom permission checks), a sensor also holds a maximum authorization Bloom permission. This permission is used to prove authorization on any requests the sensor may make. As an illustrative example, a sensor holding permission $\mathcal{P}(\lceil(\{\texttt{medic}, \texttt{fire}, \texttt{police}\}, \{\texttt{rd}, \texttt{fwd}\})\rceil)$ can request its message to be passed by a fire department sensor, and computes and presents the corresponding Bloom permission for $(\{\texttt{fire}\}, \{\texttt{fwd}\})$ to the target sensor.

A similar mechanism can be used to delegate authority between sensors. For example, a medic sensor can delegate authority $(\{police\}, \{fwd\})$ to a fire department sensor to enable its messages to be further propagated by police sensors. In this case a medic sensor can (efficiently) compute its new maximum authorization as the union of the Bloom permission $\mathcal{P}(\lceil(\{\texttt{police}\}, \{\texttt{fwd}\})\rceil)$ with its current maximum.

The proposed mechanism requires the requester to prove to the sensor that it holds a secret (Bloom permission) credential which is effectively an authentication check. Directly revealing the secret credential over a public connection is subject to a replay attack. Furthermore, in the case of an authorization check, the presenting participant needs to be sure that the recipient is entitled to receive the permission, that is, that the recipient also holds the permission.

7.5.1 Discussion

It is worth noting that using our scheme we are giving up some security to improve efficiency. Its overall security is less than that provided by an asymmetric cryptography based system using, for example, authorization certificates. On the other hand, we are assuming that system's communications are secure, that is, a third party cannot snort an access request with a Bloom permission and perform a repetition attack. To solve this last issue we use several strategies, amongst them:

1. If the application is closed, we can assume that communications are secure (secret and authentic). For example, using symmetric cryptography with pre-shared keys. Also consider that MCI applications usually

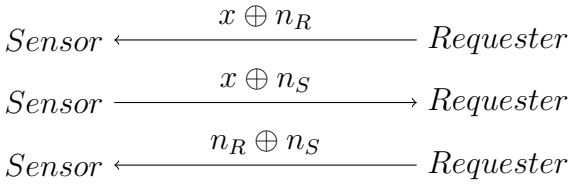


Figure 7.3: Access request protocol example.

have a short lifespan, thus reducing the possibility of compromising the keys.

2. Prevent the inclusion of the Bloom permission in sensor access requests. If we consider the Bloom permission as x, we can use a protocol where the secret key is generated by both participants (Figure 7.3).

7.6 Summary and Conclusions

In this chapter we have presented the use of hash function based permission structures (Bloom filters) as an authorization scheme in environments where asymmetric cryptography is too expensive.

We have seen that with Bloom filters, which require only simple hash calculations, we can get an acceptable level of security in networks with very low power resources as are WSNs. All of this theoretically without increasing the application's size to the point of not being usable in our very low power, both computationally and energetically, devices, and without delaying the computation time in the device drastically.

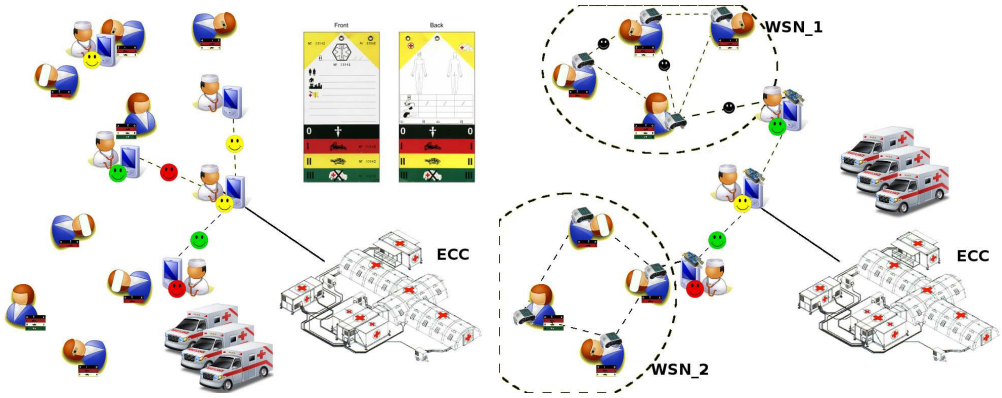
Part III

8

Conclusions and future work

IN Chapter 1 we have presented the objectives of this thesis, being its main goal to provide a secure system to automatically monitor victims in real time in emergency scenarios, even though it could be used to monitor other elements in any scenario with networking difficulties. In this chapter we discuss how we have fulfilled the foreseen objectives and present future lines of research on this topic, on some of which we have already started working on.

We started with an existing victim triaging system from our group, Mobile Agent Electronic Triage Tag (MAETT) [MRMCC09], which added a first layer of networking to victim triaging. An early form of Delay and Disruption Tolerant Networks (DTNs) were used to carry manually collected data to the ECCs. In figure 8.1a we can see a diagram depicting MAETT's actors and their role in an emergency. Victims triaged using the upper-right corner cardboard tag, medics equipped with handheld devices to create and send triage data through the network, the ECCs, destination of the data, and ambulances ready to collect already triaged victims. In this system, changes on victims' health, be them an improvement or a receding, are not recorded and may lead to erroneous or inaccurate victim collection.



(a) MAETT, our starting point (b) After our first contribution

Figure 8.1: MAETT, before and after.

In the first contribution of this thesis (Chapters 3 and 4) we propose an automated monitoring system using MAs and low-power sensor nodes to provide real-time monitoring to MAETT. There, we use the flexibility of MA to autonomously take decisions depending on their context to define a robust and low memory impact **itinerary** structure to visit every node of a WSN and read the sensor readings to finally drop this data to a passing by handheld equipped MAETT-like medic who will route the updated information to the ECC, which will handle this updates and respond accordingly. We also equip this itinerary structure with fault-tolerance methods in case one or more nodes fail or go missing during the monitoring. In this first chapters we see how this itinerary is calculated from the handheld in network building time and how it is integrated very lightly in Agilla mobile agents and further injected to the victim's WSN.

Figure 8.1b shows both the new and the old elements of the scenario working together to provide an automated victim monitoring system. Virtual WSNs are encircled and in one of them we can see little black dots representing agents sharing victim information inside the network. Colored smileys in the outer part of the virtual WSNs are the agents in charge of carrying the information collected in the WSN to the ECC in the lower right corner.

Simulations and tests done with our implementation both in closed testbeds and in the open air proved the correct working of our system with good WSN roundtrip times and fault tolerance support (See section 4.4.2 for tables and figures).

As future work regarding the agent itineraries, seen in chapter 4, it would be interesting to port our application to other WSN mobile agent middlewares such as WISEMAN or MAPS, to allow a direct comparison among those middlewares. Translating our application to a TinyOS (non-agent) environment (similar to [GMS+07]) could also be interesting to get an additional understanding of benefits and drawbacks of using mobile agents in WSNs.

As we were researching agent itineraries we knew that our networks will be limited to 32 nodes, mainly for two different matters: 1. The sensors' weight and the willingness of paramedics; and 2. Sensors' memory limits. A decision that we later found out was very accurate given the limitations of the simulator. Thus, we were aware that we would have to use more than one network to fulfill the needs of a real MCI. Our second contribution deals precisely with this matter, and we defined a clustered architecture to increase the number of monitored victim in an emergency to more than 8000 (See Chapter 5).

In our case WSN clusters were easily defined by allowing paramedics to just add sensors to their open network at a time. Then it was just a matter of increasing the number of bits used to identify a node to include the cluster identifier. Of course, the itinerary structure defined in the previous chapter had to be redefined to include the cluster identifier. Fault tolerance methods had also been modified to deal with this clustered implementation. Fortunately, the interface between WSNs and the medics' network had not had to be modified as we designed it to accept any kind of data sent from a WSN's node.

Simulations and tests done with our implementation of this clustered architecture, both in testbeds and in the open air, proved that our architecture performs well and agents in different clusters do not interfere with others. Roundtrip times were of course a bit higher than with the non-clustered approach, but were still appropriate for the monitoring needs (See section 5.2 for tables and figures).

As future work it would be interesting to improve fault tolerance methods to assure every node is frequently visited despite using lots of **random-jump** fault tolerance strategies. Also it would be interesting to compare one traveler agent results with networks with more than one mobile agent traveling the network, both with different itineraries and maybe only partial.

Once our architecture for MCI victims monitoring was finished we though

that it would be interesting to have the possibility to ask a remote specialized medic to come to another medic's position if the victim that is being triaged or treated requires special attention from an specialist. Our next contribution, described in detail in Chapter 6 handles precisely this situation.

In this third proposal, we start with an unknown network of victims and, with the inclusion of some management logic to few strategically selected nodes we can easily create a route from every node to every other node, that mobile agents can easily follow (See section 6.2.2 for the details of route generation). Basically we look for the shortest path to the special nodes and every other node uses this as their starting point ask this special nodes for the route to a particular node. A simplified form of directed diffusion is used to generate this routes.

During the simulations our agents (refer to Section 6.3 for a description of the agents involved in this system) performed quite well in small networks. Increasing their size caused a series of memory overflow errors in simulation configuration time, which we believe are caused by the simulator inability to directly inject agents to a specific node, but it has to be done always through the node 0, which has to be unavoidably directly connected to the target node.

We really believe that if we should be able to simulate our system with real nodes or in a simulator supporting a direct injection alternative, the obtained results would be interesting, making the deployment of the system an attractive alternative.

Future work regarding the retrieval of remote systems would include, of course, the simulation in larger networks of real nodes, testing other injection and agent deployment alternatives and, the design of a remote service retrieval system in clustered WSNs. Designing this second system, will kill two birds with one stone: **1**. Will enable clustered WSNs to retrieve remote services, and **2**. Will solve the problem of memory overflows when working with large networks.

A possible approach to the design of such system could be one similar to existing routing protocols in IP networks, where routers, in our case `clever` node, know where to send requests to external addresses directing them to a neighbor router who owns the addresses or knows a path to them.

As we can see in Chapter 7 we also took the time to think about security in our victim monitoring system. As we are dealing with valuable and critical

health status information we do not want it to be manipulated, modified or removed in any way, as it will worsen our victim monitoring system.

When designing this protection scheme we wanted it to be simple, lightweight and fast, first because we want it to run in very low power devices, and second because we don't want the user wait for the encryption to end. We used a known structure known as Bloom filters [Blo70] to provide a reasonable level of protection without compromising the efficiency of the system. We presented a theoretical approach to an access control system using one way cryptographic hash functions which can be used in low power devices such as our sensor nodes. The system includes permission delegation from a higher authority allowing discrete permissions to be assigned to lower members in the scheme.

Future work for this access control scheme will start with implementing the scheme in Agilla and test it both in the simulator and on actual nodes.

Finally, to conclude the conclusions ... We have built an heterogeneous system using different MA technologies and network types, working fluently together to offer a secure, flexible, fault tolerant and autonomous real-time victim monitoring system. We have presented our results for each of the parts of the system and shown the conclusions drawn in each of them. Also, we introduced several topics to further research for each of the contributions which we think will make the system even better.

Although we achieved good results with MAs in WSNs we believe that it is still in its early stages. The stability of the platform, both in real devices and in the simulator made the testing and debugging of our algorithms and architectures last longer than expected due to random errors caused by unexpected simulator crashes. What is more, successful tests in the simulator crashed then when moved to real nodes, giving us unexpected headaches and hair pulling situations, fortunately most of them solved favorably.

We strongly believe in mobile agents as an activating technology for rich and complex WSNs applications, even the other way around, as WSNs and their communication particularities create the perfect environment for MAs to develop perfectly. But it won't be until a lot of work is done in building a complete and simple middleware to be used outside the research institutions.

Part IV

A

Acronyms

WSN Wireless Sensor Network 75

DTN Delay and Disruption Tolerant Network 87

MA Mobile Agent 76

MAETT Mobile Agent Electronic Triage Tag 87

MANET Mobile Ad-hoc NETwork 37

MCI Mass Casualty Incident 76

ECC Emergency Coordination Center 77

START Simple Triage And Rapid Treatment 35

MTS Manchester Triage System 35

B

Bibliography

[AFGG11] Francesco Aiello, Giancarlo Fortino, Raffaele Gravina, and Antonio Guerrieri. A java-based agent platform for programming wireless sensor networks. *The Computer Journal*, 54(3):439–454, 2011.

[Ash09] Kevin Ashton. That 'internet of things' thing. *RFiD Journal*, 22:97–114, 2009.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 164–173. IEEE, 1996.

[BGJL06] John Burgess, Brian Gallagher, David Jensen, and Brian Neil Levine. Maxprop: Routing for vehicle-based disruption-tolerant networks. In *Proc. ieee infocom*, volume 6, pages 1–11. Barcelona, Spain, 2006.

[Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[BLV07] Aruna Balasubramanian, Brian Levine, and Arun Venkataramani. DTN routing as a resource allocation problem. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 373–384. ACM, 2007.

[BPR99] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE–A FIPA-compliant agent framework. In *Proceedings of PAAM*, volume 99, page 33. London, 1999.

[Bra91] H. Braun. On solving travelling salesman problems by genetic algorithms. In Hans Paul Schwefel and Reinhart Männer, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 129–133. Springer Berlin / Heidelberg, 1991. 10.1007/BFb0029743.

[CAM07] J. Cucurull, J. Ametller, and R. Martí. Agent mobility. In *Developing multi-agent systems with JADE*, pages 115–130. Wiley Inc., 2007.

[CCH+11] Wei Cai, Min Chen, Takahiro Hara, Lei Shu, and Taekyoung Kwon. A genetic algorithm approach to multi-agent itinerary planning in wireless sensor networks. *Mobile Networks and Applications*, 16(6):782–793, 2011.

[CGL07] Min Chen, Sergio Gonzalez, and Victor CM Leung. Applications and design issues for mobile agents in wireless sensor networks. *Wireless Communications, IEEE*, 14(6):20–26, 2007.

[con] Contiki the Open Source Operating System for the Internet of Things. `http://www.contiki-os.org`. Last Accessed: 2013-04-10.

[CYK+11] Min Chen, Laurence T Yang, Taekyoung Kwon, Liang Zhou, and Minho Jo. Itinerary planning for energy-efficient agent communications in wireless sensor networks. *Vehicular Technology, IEEE Transactions on*, 60(7):3290–3299, 2011.

[DGV04] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462. IEEE, 2004.

[DR07] Raheleh B Dilmaghani and Ramesh R Rao. A reliable wireless mesh infrastructure deployment at crisis site. In *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa*, pages 579–581. IEEE, 2007.

[EFL+99] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. Technical report, IETF RFC 2693, September, 1999.

[Fip02] ACL Fipa. FIPA ACL Message Structure Specification. *Foundation for Intelligent Physical Agents, http://www. fipa. org/specs/fipa00061/SC00061G. html (30.6. 2004)*, 2002.

[FIP04] March FIPA. FIPA agent management specification. *Standard component SC00023K, Foundation for Intelligent Physical Agents*, 2, 2004.

[FL05] Emory A Fry and Leslie A Lenert. Mascal: Rfid tracking of patients, staff and equipment to enhance hospital response to mass casualty events. In *AMIA Annual Symposium Proceedings*, volume 2005, page 261. American Medical Informatics Association, 2005.

[Fol03] Simon N Foley. Using trust management to support transferable hash-based micropayments. In *Financial Cryptography*, pages 1–14. Springer, 2003.

[FRL05] Chien-Liang Fok, G-C Roman, and Chenyang Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 653–662. IEEE, 2005.

[FRL09] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(3):16, 2009.

[GB06] Dimitrios Georgoulas and Keith Blow. In-motes: an intelligent agent based middleware for wireless sensor networks. In *Proceedings of the 5th WSEAS International Conference on Application of Electrical Engineering*, pages 225–231, 2006.

[GCB06] Miguel Escrivá Gregori, Javier Palanca Cámara, and Gustavo Aranda Bada. A jabber-based multi-agent system platform. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1282–1284. ACM, 2006.

[Gel85] David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985.

[GMS+07] Tia Gao, Tammara Massey, Leo Selavo, David Crawford, Borrong Chen, Konrad Lorincz, Victor Shnayder, Logan Hauenstein, Foad Dabiri, James Jeng, et al. The advanced health and disaster aid network: A light-weight wireless medical system for triage. *Biomedical Circuits and Systems, IEEE Transactions on*, 1(3):203–216, 2007.

[GRB08] Carles Garrigues, Sergi Robles, and Joan Borrell. Securing dynamic itineraries for mobile agent applications. *Journal of Network and Computer Applications*, 31(4):487–508, 2008.

[GVCL10] Sergio González-Valenzuela, Min Chen, and Victor CM Leung. Programmable middleware for wireless sensor networks applications using mobile agents. *Mobile Networks and Applications*, 15(6):853–865, 2010.

[GVVL06] Sergio González-Valenzuela, Son Vuong, and Victor Leung. A mobile code platform for distributed task control in wireless sensor networks. In *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, pages 83–86. ACM, 2006.

[Hen06] James Hendler (Editor in Chief). Special Issue: Intelligent Agents in Healthcare. *IEEE Intelligent Systems*, 21(6), 2006.

[Hew77] Carl Hewitt. Viewing control structures as patterns of passing messages. *Artificial intelligence*, 8(3):323–364, 1977.

[HKH09] Benjamin Hirsch, Thomas Konnerth, and Axel Heßler. Merging agents and services—the JIAC agent platform. In *Multi-Agent Programming:*, pages 159–185. Springer, 2009.

[HSW+00] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *ACM SIGOPS operating systems review*, volume 34, pages 93–104. ACM, 2000.

[IGE+03] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *Networking, IEEE/ACM Transactions on*, 11(1):2–16, 2003.

[IMM+06] KOJI IDOGUCHI, YASUMITSU MIZOBATA, TETSUYA MATSUOKA, YASUAKI MIZUSHIMA, KAZUO ISHIKAWA, HITOSHI YAMAMURA, and JUN'ICHIRO YOKOTA. Usefulness of our proposed format of triage tag. *Journal of Japanese Association for Acute Medicine*, 17(5):183–191, 2006.

[IPP+10] S. S. Iyengar, N. Parameshwaran, V. V. Phoha, N. Balakrishnan, and C. D. Okoye. *Fundamentals of Sensor Network Programming: Applications and Technology*. Wiley-IEEE Press, 2010.

[ISOF06] Sozo Inoue, Akihiko Sonoda, Kenichiro Oka, and Shinichiro Fujisaki. Emergency healthcare support: RFID-based massive injured people management. In *Proceedings of the fourth International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications, Irvine, CA*, 2006.

[jad] Java Agent DEvelopment Framework. `http://jade.cselt.it`. Last version (4.3.0 is from March 29th, 2013. Last accessed: 2013-04-17.

[JMB+01] David B Johnson, David A Maltz, Josh Broch, et al. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. *Ad hoc networking*, 5:139–172, 2001.

[JS89] K.A. De Jong and W.M. Spears. Using genetic algorithms to solve NP-complete problems, 1989.

[KKRF12] J. Kenyeres, M. Kenyeres, M. Rupp, and P. Farkas. An algorithm for central point estimation in WSNs. In *Telecommunications and Signal Processing (TSP), 2012 35th International Conference on*, pages 32–36. IEEE, 2012.

[KSMA06] YoungMin Kwon, Sameer Sundresh, Kirill Mechitov, and Gul Agha. ActorNet: An actor platform for wireless sensor networks. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1297–1300. ACM, 2006.

[LABW92] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems (TOCS)*, 10(4):265–310, 1992.

[LC02] Philip Levis and David Culler. Maté: A tiny virtual machine for sensor networks. In *ACM Sigplan Notices*, volume 37, pages 85–95. ACM, 2002.

[LDS04] Anders Lindgren, Avri Doria, and Olov Schelen. Probabilistic routing in intermittently connected networks. In *Service Assurance with Partial and Intermittent Resources*, pages 239–254. Springer, 2004.

[LM96] William Li and David G Messerschmitt. Java-to-go. *University of California, Berkeley, `http://ptolemy.eecs.berkely.edu/dgm/javatools/java-to-go`*, 1996.

[LnKM+99] P. Larrañaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170, 1999. 10.1023/A:1006529012972.

[LOKK97] Danny B Lange, Mitsuru Oshima, Günter Karjoth, and Kazuya Kosaka. Aglets: Programming mobile agents in Java. In *Worldwide Computing and Its Applications*, pages 253–266. Springer, 1997.

[Mas05] D. Massaguer. Multi mobile agent deployment in wireless sensor networks. Master's thesis, University of California, Irvine, 2005.

[MB03] Joan Mir and Joan Borrell. Protecting mobile agent itineraries. In *Mobile Agents for Telecommunication Applications*, pages 275–285. Springer, 2003.

[MFV+06] Daniel Massaguer, Chien-Liang Fok, Nalini Venkatasubramanian, Gruia-Catalin Roman, and Chenyang Lu. Exploring sensor networks using mobile agents. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 323–325. ACM, 2006.

[MJMW06] Kevin Mackway-Jones, Janet Marsden, and Jill Windle. *Emergency triage*. BMJ Books, 2006.

[MLC98] Dejan S Milojičić, William LaForge, and Deepika Chauhan. Mobile objects and agents (MOA). In *Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems-Volume 4*, pages 13–13. USENIX Association, 1998.

[MNAFB12] Estanislao Mercadal, Guillermo Navarro-Arribas, Simon N Foley, and Joan Borrell. Towards efficient access control in a mobile agent based wireless sensor network. In *7th International Conference on Risk and Security of Internet and Systems (CRiSIS), 2012*, pages 1–4. IEEE, 2012.

[MRM+12] Estanislao Mercadal, Sergi Robles, Ramon Martí, Cormac J Sreenan, and Joan Borrell. Double multiagent architecture for dynamic triage of victims in emergency scenarios. *Progress in Artificial Intelligence*, 1(2):183–191, 2012.

[MRMCC09] R Martí, S Robles, A Martín-Campillo, and J Cucurull. Providing early resource allocation during emergencies: The mobile triage tag. *Journal of Network and Computer Applications*, 32(6):1167–1182, 2009.

[MVSB13] Estanislao Mercadal, Carlos Vidueira, Cormac J. Sreenan, and Joan Borrell. Improving the dynamism of mobile agent applications in wireless sensor networks through separate itineraries. *Computer Communications*, 36(9):1011 – 1023, 2013.

[nas] Distruption Tolerant Networking for Space Operations (DTN). `http://www.nasa.gov/mission_pages/station/research/experiments/730.html`. Last Accessed: 2013-04-10.

[NG99] Antony Nocera and Alan Garner. Australian disaster triage: a colour maze in the tower of babel. *Australian and New Zealand journal of surgery*, 69(8):598–602, 1999.

[NS10] D Michael Navin and William J Sacco. Science and evidence-based considerations for fulfilling the salt triage framework. *Disaster Medicine and Public Health Preparedness*, 4(01):10–12, 2010.

[ON98] Paul D O'Brien and Richard C Nicol. FIPA—towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.

[Pap77] C.H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237 – 244, 1977.

[Ped97] Torben P Pedersen. Electronic payments of small amounts. In *Security Protocols*, pages 59–68. Springer, 1997.

[PR99] Charles E Perkins and Elizabeth M Royer. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*, pages 90–100. IEEE, 1999.

[PSC05] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364–369. IEEE, 2005.

[QW01] Hairong Qi and Feiyi Wang. Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks. *Proceedings of the IEEE*, 2001.

[RS97] Ronald L Rivest and Adi Shamir. Payword and micromint: Two simple micropayment schemes. In *Security Protocols*, pages 69–87. Springer, 1997.

[SBH96] Markus Straßer, Joachim Baumann, and Fritz Hohl. Mole-a java based mobile agent system. In *Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems*, pages 28–35, 1996.

[Sen10] Jaydip Sen. A survey on wireless sensor network security. *arXiv preprint arXiv:1011.1529*, 2010.

[SLO05] Leo Szumel, Jason LeBrun, and John D Owens. Towards a mobile agent framework for sensor networks. In *Second IEEE Workshop on Embedded Networked Sensors*, pages 79–87, 2005.

[SPR05] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259. ACM, 2005.

[SRJB03] R.C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling a three-tier architecture for sparse sensor networks. In *Proc. Sensor Network Protocols and Applications (SNPA)*, pages 30 – 41. IEEE, May 2003.

[SRM+98] Markus Stra, Kurt Rothermel, Christian Maihöfer, et al. *Providing reliable agents for electronic commerce*. Springer, 1998.

[ST98] Tomas Sander and Christian F Tschudin. Protecting mobile agents against malicious hosts. In *Mobile agents and security*, pages 44–60. Springer, 1998.

[Sup84] G Super. Start: a triage training module. *Newport Beach, CA: Hoag Memorial Hospital Presbyterian*, 1984.

[tin] TinyOS Home Page. `http://www.tinyos.net`. Last Accessed: 2013-04-10.

[VB+00] Amin Vahdat, David Becker, et al. Epidemic routing for partially connected ad hoc networks. Technical report, Technical Report CS-200006, Duke University, 2000.

[VGB08] Christian Vecchiola, Alberto Grosso, and Antonio Boccalatte. AgentService: a framework to develop distributed multiagent systems. *International Journal of Agent-Oriented Software Engineering*, 2(3):290–323, 2008.

[Vig97] Giovanni Vigna. Protecting mobile agents through tracing. In *Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, Jyvälskylä, Finland*. Citeseer, 1997.

[WCKC11] X Wang, M Chen, T Kwon, and HC Chao. Multiple mobile agents' itinerary planning in wireless sensor networks: survey and evaluation. *Communications, IET*, 5(12):1769–1776, 2011.

[Whi99] James E White. Telescript technology: mobile agent. In *Mobility*, pages 460–493. ACM Press/Addison-Wesley Publishing Co., 1999.

[WPW+97] David Wong, Noemi Paciorek, Tom Walsh, Joe DiCelie, Mike Young, and Bill Peet. Concordia: An infrastructure for collaborating mobile agents. In *Mobile Agents*, pages 86–97. Springer, 1997.

[WRB+04a] Q. Wu, N.S.V. Rao, J. Barhen, S.S. Iyengar, V.K. Vaishnavi, H. Qi, and K. Chakrabarty. On computing mobile agent routes for data fusion in distributed sensor networks. *IEEE Trans. on Knowl. and Data Eng.*, 16(6):740–753, 2004.

[WRB+04b] Qishi Wu, Nageswara SV Rao, Jacob Barhen, SS Iyenger, Vijay K Vaishnavi, Hairong Qi, and Krishnendu Chakrabarty. On computing mobile agent routes for data fusion in distributed sensor networks. *Knowledge and Data Engineering, IEEE Transactions on*, 16(6):740–753, 2004.

Estanislao Mercadal Melià
Bellaterra, September 2013