Universitat Autònoma de Barcelona
Departament d'Enginyeria de la Informació i de les Comunicacions

A Mobile Code-based Multi-Routing Protocol Architecture for Delay and Disruption Tolerant Networking

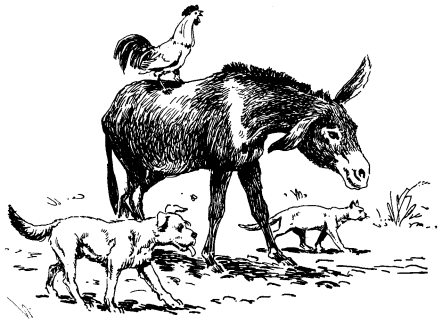
Carlos Borrego Iglesias

Bellaterra, January 2013

Advisor Dr. Sergi Robles

UAB

Universitat Autònoma de Barcelona
Departament d'Enginyeria de la Informació i de les Comunicacions



A Mobile Code-based Multi-Routing Protocol Architecture for Delay and Disruption Tolerant Networking

Carlos Borrego Iglesias

Bellaterra, January 2013

Advisor Dr. Sergi Robles

UAB

LaTeXstyle:

http://abra.uab.es/∼cborrego/fancy-phd.sty

Front Cover Photography:

http://www.carlosborregoiglesias.com

Title Page Illustration:

http://www.arthursclipart.org

Printed in:

La casa de l'atzar

I certify that I have read this thesis entitled "A Mobile Code-based Multi-Routing Protocol Architecture for Delay and Disruption Tolerant Networking" 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, January 2013

Dr. Sergi Robles
(Advisor)

Committee:

Dr. Joan Borrell
Dr. Josep Pegueroles
Dr. Cormac Sreenan
Dr. Jordi Herrera (substitute)
Dr. Ricardo João Cruz Correia (substitute)

Program: *Doctor en Informàtica.*

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

A mis padres y a mi hermana

Abstract

IN THIS THESIS, we include code in the messages to improve Delay and Disruption Tolerant Networking (DTN) performance. Store-carry-and-forward DTN protocols offer new possibilities in scenarios where there are intermittent connectivity, asymmetric bandwidths, long and variable latency and ambiguous mobility patterns. However, there are scenarios where current DTN mechanisms are not efficient enough, for example when several applications need to coexist. In this study we present a general purpose architecture for this type of scenarios based upon the idea of letting the application, by means of its messages, decide the behavior of every intermediate node. The keystone of this proposal is to carry the routing algorithm code along with every single message. The resulting DTN can be used by different heterogeneous applications simultaneously. This thesis uses three examples of three different scenarios to show how our architecture can be used. Firstly, it presents a general purpose, multi-application mobile node sensor network based on mobile code. This intelligent system can work in DTN scenarios. Mobile nodes host software mobile code with task missions and act as DTN routers following the store-carry-and-forward model. Secondly, it introduces a new paradigm – *store-carry-process-and-forward* – based on mobile code to improve the integration of wireless sensor networks and grid computing infrastructures. Thirdly, it describes an emergency scenario in which different users such as policemen, firemen, doctors, paramedics, engineers or rescue teams, among others, along with portable devices such as smart phones or tablets, create the interconnected network. Opportunistic contacts among the different users permit the different applications to employ the network for very different purposes. Additionally, the proposal is complemented by an integration based on code block bundle extensions of the proposed architecture with the DTN Bundle Protocol. The feasibility and usability of the different application proposals are proved and evaluated by comparing its performance with state-of-the-art proposals.

Acknowledgements

THIS THESIS would not have been the same without the valuable help and support from a lot of people. First of all, I would like to thank my family for their continuous support throughout my education. Muchas, muchas gracias de corazón, familia. Thanks heaps to Gabriela Ellena and Isabel Coderch for those first days of hilarious presentation rehearsals. Those were the days! Thanks to Jordi Requena and Gonçalo Pinto for his continuous interest. Thanks to Judy Kaufmann and Clara Juan for all your design tips. Thanks to my university friends in Madrid for all these years of non-computer science conversations.

I am very thankful to Simin Nadjm-Tehrani and the rest of the RSTLAB laboratory at Linköping University for the warm welcome I received while staying in Sweden during the summer of 2011. *Grazie di cuore* to Andrei Maslenikov, my supervisor during my stay in Rome, for being such a generous and stimulating supervisor, freely permitting me to find my path. *Gracias* to Sergio Arévalo, my final thesis supervisor at my home university which encouraged me to apply to a CERN job. *Merci bien* to Christian Boissat at CERN for being such a inspiring supervisor.

Gràcies to everybody in the dEIC department, specially to Jordi Cucurull who during my first days of mobile agent research helped me a lot. Working around such a talented group of people in such a pleasant atmosphere has been very enriching. Thanks so much as well to Jordi Nadal, Arnau Bria and Esther Acción from PIC, my first job in the Autonomous University of Barcelona.

I do not have enough words to express how thankful I am to Sergi Robles, my thesis supervisor. My time spent while developing this thesis has always been very gratifying. Thanks for all the never-ending conversations about grammar, etymology, science, politics, philosophy and from time to time computer networks. Thanks as well for giving me the opportunity of teaching in the dEIC department. Now I know what it is to adore a job.

My last and very special and warm thank you to Andreu Pacheco, my supervisor while working for the IFAE institute. I would have never started my thesis without his help. I really appreciate his support all these last three years. His kindness has been an inspiration to me. The people that work with you are very lucky.

This work has been partially supported by the Spanish Government research projects FPA2010-21919-C03-02 from the *Ministerio de Economía y Competitividad* and TIN2010-15764 from the *Ministerio de Ciencia e Innovación*. I am very thankful to all these government institutions, but very concerned as well for those who in a near future will not have the same economical support as me for starting or finishing their research studies.

Carlos Borrego Iglesias
Barcelona, January 2013

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Preliminaries

"In what could be seen as a paradox, latest research studies revolve around poorly connected networks. These networks were already present when the growth of the Internet was beginning to arise; however, the research-worthy requirements for these networks were met once communication devices were revolutionized."

"Mobilis in mobili.", Twenty Thousand Leagues Under the Sea

Jules Verne

"Although this may seem a paradox, all exact science is dominated by the idea of approximation. When a man tells you that he knows the exact truth about anything, you are safe in inferring that he is an inexact man."

Bertrand Russell

1 Introduction

WE LIVE in interesting times, for the good and the bad. The Internet has proven to be an infrastructure capable of interconnecting different types of networks all along the globe. The aim for this global network is to provide a large variety of applications worldwide.

While still in constant evolution, the almost newborn networks have thrillingly changed the way we socialise and interact. Everyday living activities are more and more related to network communications. The widespread of embedded network devices such as mobile phones, tablets, laptops, etc. in conjunction with the deployment of wireless networks such as Wi-Fi, GSM or Bluetooth, physically bring permanent network connection closer to scenarios where traditionally it would have been very difficult to be deployed.

The majority of the available applications take for granted a permanent connection among the different parties involved in the communication. Datagrams traveling through these networks do not know in advance their path to the destination, however, it is assumed that a continuous path from the source to the destination will be found.

In the hyperconnectivity era, there is room for other types of person-to-person, machine-to-machine and person-to-machine communications. Scenarios which lack continuous network connectivity appear to be an interesting topic for the research community. Terms like Delay and Disruption Tolerant Networking (DTN), opportunistic networks, challenged networks, partitioned networks, etc. arise in the research community for solving communication problems and offer promising networks solutions for novel intermittently connected scenarios. The result is the possibility of running applications in challenged scenarios where under different circumstances using classical network protocols would have been impossible.

In what could be seen as a paradox, the latest research studies revolve around poorly connected networks. These networks were already present when the growth of the Internet was beginning to arise; however, the research-worthy requirements for these networks were met once communication devices were revolutionised.

Among the different topics around these opportunistic networks, the most effortful and interesting one is routing. The routing problem is a matter of making a decision with the challenge that useful information for making optimal decisions may be not be available. New paradigms for making these decisions have been proposed, and this thesis is one of them, but with a particular perspective.

One of the greatest lessons learned over these years in the study of how to improve Delay and Disruption Tolerant Networks is that, unlike traditional networks such as the Internet using protocols like TCP/IP, a unique solution to cover all the possible scenarios and applications is unachievable. DTN scenarios are far different from one another and depend on a wide range of different characteristics which will described along this thesis.

Considering the evolution of other networks such as the Internet in which different applications efficiently employ a network infrastructure, in this thesis we analyze the problem of application coexistence in Delay and Disruption Tolerant Networking scenarios. We strongly believe in the benefits of taking advantage of the different intermittently connected network devices belonging to different applications in order to allow several applications to simultaneously use the very same network. However, if several different applications coexist, the best routing algorithm for them all is not a trivial matter. These and other issues involved application coexistence have to be considered and can be very hard to solve.

Our proposal will always be introduced as a solution for certain DTN scenarios. Our goal for this study is far from a general solution to solve all DTN problems, instead some very useful procedures and mechanisms for solving DTN problems such as routing, multi application coexistence, context awareness, network congestion, flow control, etc. are presented for different scenarios.

Having the opportunity to do research on this subject has been a gratifying experience. My personal PhD path has been quite different from traditional ones: I started once I had been working for some years abroad and I have been combining my research with a job in a particle physics research institute. I totally uphold this PhD route (a truer word was never uttered). The experience has been very enriching and educative thus far but not without its challenges.

Unfortunately, not everything around developing a thesis is as fascinating as the research itself. The acclaimed process of article evaluation or peer reviewing in which qualified experts within a certain science field are involved may have some drawbacks. I have been suffering from poor reviews from journals which after almost one year of manuscript reviewing show that beyond not understanding our proposals, give their absolute lack of knowledge away. Other proposals for manuscript reviewing for improving reviewing time and accuracy reviewing could be considered for the sake of research improvement. Some studies like [105], bring under the spotlight the necessity of alternative mechanisms to the peer-review. In a meaningless effort of doing one's bit, I highly support a change for manuscripts reviewing.

Fortunately, I have managed to publish some articles in international conferences and international journals. Going through this process has been very hard and has provided me with nothing but frustration. During these years I would have preferred spending less time on futile details evolving journal publishing and more time on my research itself. Another research paradigm is possible.

1.1 Objectives

The objective of this thesis is to provide different mechanisms to improve the coexistence of several applications in Delay and Disruption Tolerant Network scenarios. The keystone of these mechanisms is the possibility of carrying the routing algorithm code along with the data messages themselves. This new paradigm offers new possibilities to the applications employing the network. Our aim has been to study the benefits of carrying such code and some other

codes which will be described in detail throughout this thesis. These benefits have been achieved by using this mobile code in different network aspects such as application data scheduling, lifetime control, network congestion, application data aggregation, as well as to provide additional services such as message data processing or message-based mobile node movement influence. Our purpose has always been to supplement our proposal with real-case scenarios applications, as well as to study its benefits in comparison with the state of the art methods. Our aspiration has been as well to integrate our proposal with de facto DTN standard models.

1.2 Contributions

The original contributions of this thesis are the following:

- A design of a generic architecture for opportunistic networks based on mobile code.
- A design and implementation of the proposed architecture to improve intermittently connected applications in emergency scenarios.
- A design and implementation of a general purpose, multi-application mobile node sensor network based on mobile code.
- A design and implementation of a new paradigm – *store-carry-process-and-forward* – based on mobile code to improve the integration of wireless sensor networks and grid computing infrastructures.
- The implementation of a delay tolerant grid service, the computer element, to give computing access to intermittently connected wireless sensor networks.
- An integration of the proposed paradigm in the DTNRG's Bundle Protocol.
- A simulation environment to study the architecture proposal performance.

1.3 Document Layout

In the following paragraphs the outline of the parts and sections that form this thesis is presented.

The thesis is divided in five parts. Each part starts with a quote from the part itself. The first part, *Preliminaries*, includes the sections with preliminary information to understand the rest of the document. This part contains this introduction section and following, in Chapter 2, the state of the art, an introduction to the main proposal around Delay and Disruption Tolerant Networks and mobile code.

In the second part, *Proposal*, the core of the proposal is described: in Chapter 3 a novel context-aware and multi-routing protocol architecture for Delay and Disruption Networks based on mobile code is presented. In Chapter 4, three different ways of making this architecture concrete are explained. Firstly, in Section 4.1, a mobile agent-based reification is presented. Secondly, in Section 4.2, a way of using mobile agents as data and code containers is presented. Finally, in Section 4.3, a Bundle Protocol-based reification is described.

In the third part, *Scenarios*, three different scenarios in which the thesis proposal can be employed are described. These three different scenarios are implementations of the three different reifications from Chapter 4. In Chapter 5, a general purpose, multi-application mobile node sensor network based on mobile code is presented. The principle of this proposal is using mobile code at two levels: for the application and for the definition of the behavior in terms of routing algorithms, movement policies and sensor retrieval preferences. The system can adapt to the environment, dynamically optimizing routing algorithms using local and global information and influencing node movement. In Chapter 6, a new paradigm is proposed, the *store-carry-process-and-forward*, based on mobile code to improve the integration of wireless sensor networks and grid computing infrastructures. We describe the implementation of a delay tolerant grid service, the computer element, to give computing access to an intermittently connected wireless sensor network. The result is an intelligent system which adapts dynamically to intermittent disconnections and improves multi-application coexistence. In Chapter 7, a DTN disaster recovery application for calamities such as terrorist attacks or meteorological catastrophes, is described. In this scenario, doctors, nurses and rescue teams, among other users, form the Delay and Disruption Tolerant Network.

The fourth part, *Conclusions*, presents the conclusions which have been

drawn and future lines for prospect ulterior studies. Finally, in part five, *Bibliography*, the different article and book references are listed.

1.4 List of Publications

This thesis revolves around the concepts presented in the following publications from international conferences and JCR journals:

- *A store-carry-process-and-forward Paradigm for Intelligent Sensor Grids* C. Borrego, S. Robles, in Journal of Information Sciences, 2013 [11].
- *ATLAS Site Status Board. Automatic exclusion and a monitoring on ATLAS computing activities*, C. Borrego et al. In Iberian Grid Infrastructure Conference IBERGRID 2011, 2011 [13].
- *Seguridad en la planificación de agentes móviles en redes DTN* C. Borrego, S. Robles, In Reunión Española sobre Criptología y Seguridad de la Información, RECSI 2010, 2010 [14].
- *Mobile Agent Virtual Organisation to Improve Relative Information in Grid Services* C. Borrego, S. Robles. In Proceeding 3PGCIC '10 Proceedings of the 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2010 [15].
- *Relative Information in Grid Information Service and Grid Monitoring using Mobile Agents*, C. Borrego, S. Robles, In International Conference on Practical Applications of Agents and Multiagent Systems. Universidad de Salamanca, 2009 [12].
- *Distributed ATLAS computing activities* X. Espinal, C. Borrego et al., In IBERIA Proceedings of the 2nd Iberian Grid Infrastructure Conference, pg. 19-30 Porto, Portugal, May 12-14, 2008 [40].
- *File Transfer Service and CMS data transfer optimizations at PIC Tier-1 center*, J. Flix, C. Borrego, et al., EGEE 2007 Conference, Budapest, 2007 [42].
- *Control de acceso para mensajes proactivos en redes DTN* A. Sánchez, C. Borrego, S. Robles, J. Andújar, in Proceedings for the XII Reunión

Española sobre Criptología y Seguridad de la Información (RECSI 2012), 2012 [81].

- *Automating ATLAS Computing Operations using the Site Status Board* J. Andreeva, C. Borrego, et al. In Computing in High Energy and Nuclear Physics 2012, New York, NY, USA, 21 - 25 May 2012 [4].

- *Publicación de Información y Monitorización relativa usando Agentes Móviles en la Computación Grid*, Master Thesis Research Study, 2008 [10].

"If classical music is the state of the art then the arts are in a sad state."

Frank Zappa

"You have to do research. You have to go on Amazon and read a really long review written by an insane person... who's been dead for months because he shot his wife and then himself after explaining to you that the remote is counter-intuitive. It's got really small buttons on the remote he said before he murder-suicided his whole family."

Louie C.K.

2

State of the Art

THIS CHAPTER, describes different technologies, protocols and architectures to understand the rest of the thesis. On one hand, the main issues around Delay and Disruption Tolerant Networking will be described in detail. This chapter is divided into five sections. The first section introduces the concept of Delay and Disruption Tolerant Networking. The second section explains the routing problem in these types of networks. In the third section a description of the main architecture proposal for Delay and Disruption Tolerant Networking (DTN) is presented. In the fourth section, the bundle protocol is described. Finally, in the fifth section, a description of the mobile code technology, the technology on which the architecture introduced in this thesis is based, will be presented. During the chapters belonging to Part III, the Scenarios part, supplementary state of the art proposals concerning the different scenarios will be described.

2.1 Introduction to Intermittently Connected Networks

There are many projects involved with networks which are characterised by intermittent connectivity, asymmetric bandwidths, long and variable latency and ambiguous mobility patterns. The reasons for this lack of end-to-end connectivity may be due to several reasons such as restrictions on the use of devices, zone security problems or scenarios including out of range zones. These networks are generally divided into regions depending on their characteristics. The limits among these regions are defined by concepts like the delay between the links, intermittent connectivity between source and destination, the asymmetry in the speed of the connections, error rates, addressing, mechanisms of reliability, quality of service and trust relationships.

In such networks, classical Internet protocols, such as TCP/IP, cannot be directly employed due to data loss reasons, many TCP retransmissions which usually end by closing the connections. Delay and Disruption Tolerant Networking supports these communication delays by following the paradigm *store-carry-forward*: messages are sent to intermediate nodes, also known as custodian nodes, where they are kept and potentially stored for a long period of time.

In this section, on one hand, intermittently connected network routing algorithms will be defined, explained and classified. On the other hand, one of the architectures proposed for solving intermittent connected networking problems, as representative for the current state of the art technology for this research field, will be described. This proposal is the Delay Tolerant Network Research group, which has defined an end-to-end protocol, abstract service description for the exchange of what they call bundles [84] in Delay Tolerant Networking.

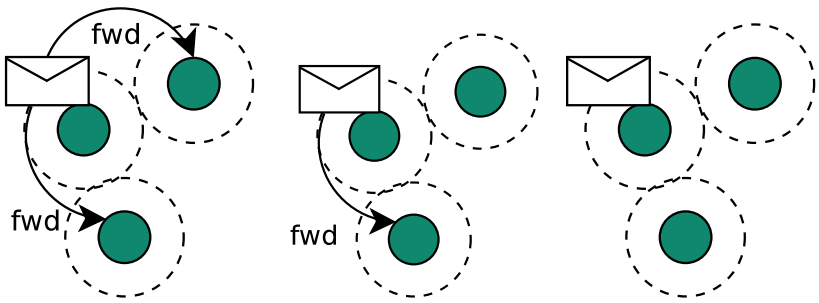
2.2 Routing in Intermittently Connected Networks

This section introduces the problem of routing in Delay and Disruption Tolerant Networking. By definition, in traditional networking, routing is a path decision process. Traditional routers in networks like the Internet choose from different routers to forward a unique copy of a datagram. In some scenarios, this forwarding approach may differ. Routing protocols in intermittently connected

networks may replicate information for delivery ratio improvement purposes. The routing algorithm needs to make decisions such as when to forward a message, where to send it, which message to send, which message to delete or which message to accept. Concerning the number of copies forwarded on each DTN node, algorithms may be:

- **Forwarding based:** A single copy of the message is kept in the network. Custodian nodes decide among the neighbours contacted the best forwarder which will become the next custodian of the information.
- **Replication based:** Insert multiple copies of a message to increase the likelihood of delivery. The algorithms must meet a compromise between the resources used and the likelihood of delivery.
- **Hybrid strategy:** combines features of the algorithms based forwarding and replication.

For example, in the Figures 2.1, a custodian node decides among three possibilities: forwarding to a given single node (Figure 2.1(a)), forwarding to a set of nodes (Figure 2.1(b)), without discarding the option of keeping a copy of the original message and finally, not forwarding the message by any means and keeping the application information (Figure 2.1(c)).



(a) Epidemic routing. (b) Unicast routing. (c) No routing.

Figure 2.1: Intermittently connected networks routing paradigms: epidemic routing, Unicast routing and no routing. Outer circles represent wireless range.

Intermittently connected networks can be subdivided into two types depending on the type of node availability: **opportunistic contacts** and **scheduled**

contacts. Networks based on opportunistic contacts are those in which communication is available only for periods which are not planned. Moreover, in scheduled contact scenarios, communication nodes move following predictable movements, which allow them to plan or schedule other nodes positions in advance for future communication purposes. For example, a bus in a region where the Internet is not widely deployed may travel from one city to another transmitting information upon arrival to the different cities following a given schedule which a priori can be foreseen. To perform this type of communication, nodes must be synchronised.

Based on the knowledge of the topology of the network routing protocols may be identified as:

- **Based on an oracle:** A node or a set of nodes have a certain knowledge of the network or its possible evolution. This knowledge includes metrics such as contacts summary, traffic status or queuing status. In terms of this knowledge the routes will be determined.

- **Model-based:** The routing algorithms use certain models to determine the routes.

- **Epidemic:** Flooding the network to all nodes with the consequent disadvantage of a high use of the custodian storage resources.

- **Based on estimation:** Based on the calculation of path probabilities to reach the destination. Messages are sent to those nodes which have a higher probability to contact the destination node.

- **Erasure Coding:** These types of routing use coding theory to establish the routes and in particular erasure codes.

- **Based on node movement control:** The routing algorithms are used to control the physical movement of nodes.

- **History-based:** Data obtained from the past is employed to efficiently route the information among the different custodian nodes.

- **Movement-based:** Information is routed taking into account the movement of the neighbour custodian nodes.

- **Beacon-less:** The majority of the routing protocols decide among the different neighbour custodian nodes where to route their information. These neighbours are discovered by means of regularly sending beacon messages to the network. However, there a few routing algorithms which do not send these messages in order to minimise the impact on the network.

- **Topological:** Routing protocols may employ information on the links, such as next-hop, bandwidth, etc... to route information.

- **Position-based:** Position information such as GPS are taken into account by routing algorithms. Parallel location services may be needed to be deployed in the network to update node positions.

Additionally, routing protocols may be classified in terms of which resource availabilities are studied. These resources include:

- **Memory.** Depending on the type of devices, memory resources can be limited. Routing algorithms are studied precisely around this limitation. Custodian buffer space management is a crucial topic in intermittently connected networks.

- **Processing.** CPU limitation is a similar problem to memory management, specially when custodian nodes are mobile devices.

- **Network bandwidth.** Network bandwidth, in conjunction with other network variables such as the network range, defines the amount of information that is able to be transmitted when two of more nodes contact. Since contact time windows may be small in DTN networks, the use of the network bandwidth is a key issue in opportunistic networking. Network bandwidth is a key issue which should be taken deeply into account when defining a routing algorithm.

- **Energy consumption.** Routing algorithms need to consider energy consumption as a limited resource, especially when dealing with mobile nodes not connected to a power line.

2.3 Delay and Disruption Tolerant Networking Architecture

A research group created by the Internet Research Task Force, (IRTF) and Delay Tolerant Networking Research Group (DTNRG) [107], is responsible for defining an architecture for Delay and Disruption's (DTN) RFCs 4838 [18]. DTNRG has defined, among others, two types of protocols called Bundle Protocol [84] and Licklider Transmission Protocol [20]. These protocols exceed the limitations and potential problems of Internet protocols such as TCP/IP.

The Bundle Protocol is described in RFC 5050 [84] paying special attention to describing the end-to-end protocol, the format of the blocks and summarise the services to exchange messages in bundles. The Licklider Transmission Protocol (LTP) is described in RFC 5326 [20] and is designed to provide reliability over links characterised by a long delay between the sending and confirmation (Round Trip Time) as well as frequently interrupted connectivity.

The layered DTN architecture proposed by the DTNRG is based on the Internet TCP/IP five layered architecture. Since DTN nodes must implement the already mentioned *store-carry-forward* paradigm, an extra layer is added to handle delays and disruption. This layer is localised as seen in Figure 2.2 between the application layer and the transport layer and it is patently called the Bundle layer. This layer enables communication across multiple nodes and regions.

The Bundle Layer stores and forwards bundles between nodes. It uses a single protocol, the Bundle Protocol, for all network regions, that form a DTN. Lower layers are chosen depending on the characteristics of each region. As in the TCP/IP architecture, each layer adds a header for encapsulation purposes, following the scheme in Figure 2.2.

Within a DTN, different devices can be present. A DTN node is an entity implementing the bundle layer. A node can be a computer, router, gateway or any combination of these which acts as source, destination or bundles forwarder.

- **Host.** Source or destination for bundles. They do not forward bundles, unless they behave as routers or gateways as well.

- **Routers.** Forward bundles within a region. Routers that operate over links with long delays need to keep bundles in a persistent storage until they can be forwarded.

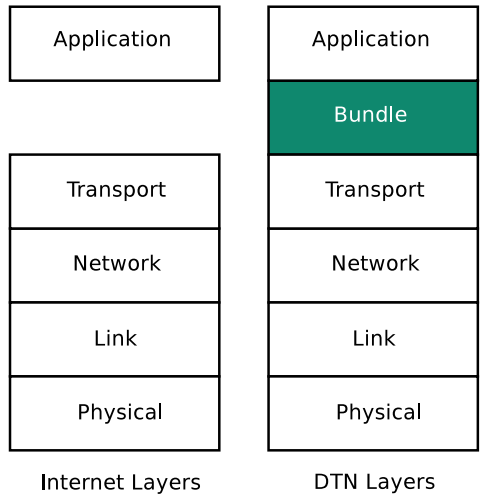


Figure 2.2: Difference among the TCP/IP network architecture and the DTN architecture.

- **Gateways.** Forward bundles from two or more DTN regions and may optionally be a router or a host. Gateways necessarily require persistent storage. Their main function is to substitute between low-level layer region protocols which are connected.

In the Internet architecture, TCP provides reliability to end-to-end communications. In the case of DTN, the bundle layer delegates on the lower layer protocols to ensure reliability in the communications. The bundle layer can be seen as a surrogate for the end-to-end layer but just for the intermediate nodes. The interface between the common Bundle Protocol and a specific inter network protocol suite is defined as convergence layer adapter.

The bundle layer supports reliability between adjacent nodes. It is done by the so-called custody transfers. When the bundle layer sends a bundle to an adjacent node, it can be requested a custody transfer so if the destination node accepts it, it will send a bundle acknowledgment. If this acknowledgment is not received before a certain time expiration, the sender will retransmit the bundle. The bundle will remain in the custody node until another node accepts its custody or the bundle lifetime expires. This mechanism does not guarantee

the end-to-end reliability and can only be done if the source of communications requests the transfer custody.

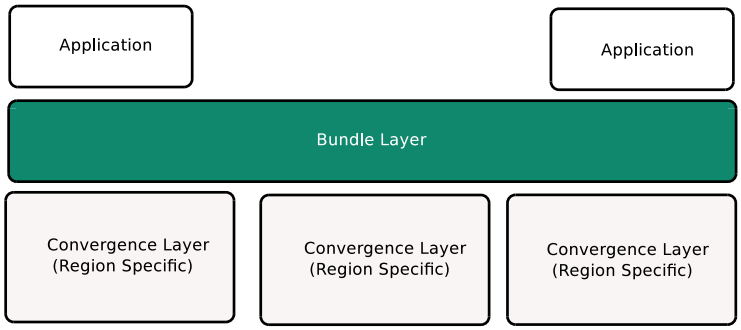


Figure 2.3: A Bundle layer is defined between the application layer and the different possible convergence layers.

2.4 The Bundle Protocol

In [84], the Bundle Protocol, a common protocol for nodes running on Delay and Disruption Tolerant Networks (DTN) is proposed. The Bundle Protocol defines a series of data blocks to route data from a source to a destination following the store-carry-and-forward paradigm: each node stores application data and whenever the node contacts another node, it may forward the application data. The bundle architecture behaves as an overlay network and additionally a naming service is defined based on Endpoint Identifiers (EIDs). Besides the source and destination endpoints of a bundle, EIDs are used to identify other endpoints that are involved in the conveyance of a bundle.

In [89], an extension meta-data block that may be used with the Bundle Protocol is defined. This block is designed to carry additional information that DTN nodes can use to make processing decisions regarding bundles. In Figure 2.4, generic fields for this extension block are depicted.

Additionally, the Bundle Protocol contains two bits to define three different types of priorities in its header. Using the previous extension meta-data block, the extended class of service described in [19] enhances the Bundle Protocol in order to allow the sender to specify additional different priorities among bundles.

These priorities, both defined in [84] and [19] are equivalent to the ones defined in the *Differentiated Services* field in the Internet Protocol Suite in which IP datagrams are ordered in different classes of precedence. While this field could force routers to act on datagrams, empirically, as described in [65], it is seldom used and it does not guarantee the priority set.

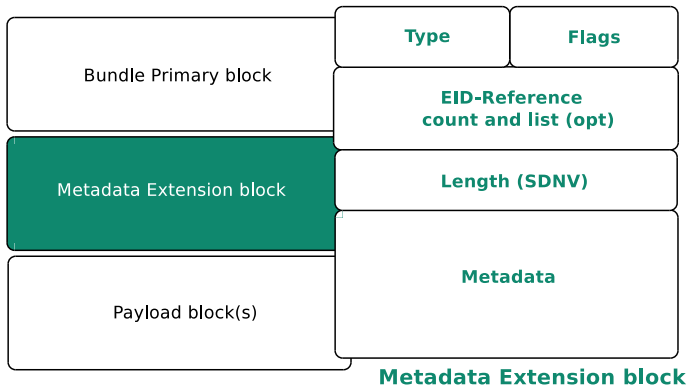


Figure 2.4: DTN Bundle blocks with extension fields detailed.

2.5 Mobile Code

Mobile code is a piece of software transferred from one machine to another and executed on the local destination machine. The basis of the proposed architecture for Delay and Disruption Tolerant Networking in this thesis is mobile code. In this thesis we will defend the fact that a possible way of facing the different issues around Delay and Disruption Tolerant Networking is by using mobile code to make, autonomously and in a context aware fashion, some of the decisions regarding routing and movement determinations. Mobile code technology [47] is a well-known procedure supporting precisely this.

In general terms, a mobile code is able to migrate from machine to machine and occasionally continue their execution on the destination machine. The entities on which mobile code run are generally called execution environments. Among the different mobile code paradigms we can distinguish two types of mobility:

- *Weak mobility* is limited to the code and the data state which is actually transferred. Before being forwarded the code updates its state so when starting from scratch it may know which was the work already performed. Most of execution environments such as Jade [6] and Agentscape [98], allow code from foreign execution environments to be executed, without keeping the state of this code. When arriving in the new execution environments, the code would be restarted.

- *Strong mobility* allows code to be moved at a very low level, such as stack pointers and instruction pointers to different execution environments. If implemented, a maximum flexibility is obtained. The code starts a thread and once it has the possibility to migrate, it suspends its code and continue running it on the destination execution environment.

  As described in [25], strong mobility code is not widely deployed. Few implementations have been done and Java is no supporting it for the moment. Strong mobility implies almost no difference among execution environments. This option is more transparent and flexible, but it is very difficult to implement.

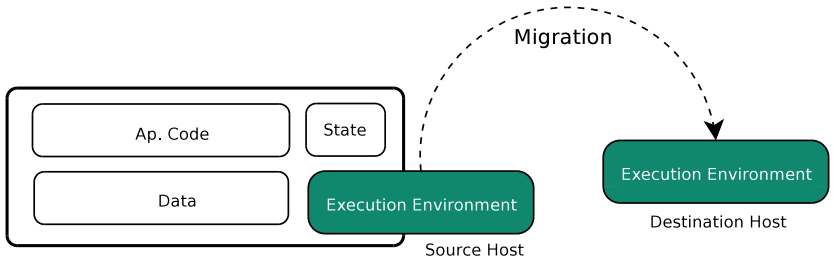


Figure 2.5: A mobile code migrates from one execution environment to another carrying its state, code and data.

There are not many publications on studies which use mobile code in the context of DTN scenarios. In [64], the idea of using mobile code which would act as *wrappers* on messages is sketched. This mobile code allows the messages to carry application information to a given destination while staying at

intermediate nodes of the network when the migration, that is, travelling to an available node, is not possible. Although they provide an interesting algorithmic approach, they give neither details about an architecture nor an implementation.

Moreover, [39] presents an interesting modelling of the movement of active messages carrying application data from fragmented wireless sensor networks. Despite the extremely detailed model proposed, no implementation and no description on how to use it in a real case application is given.

Mobile agents, software entities whose code and data migrate from host to host and resume their execution upon destination, are good candidates to implement a data-driven approach to DTN networks. Mobile agents need execution environments for running, the platforms. Jade [6], Agentscape [98] and Mobile-C [26] are popular examples of platforms supporting mobile agents. A platform can be installed on every DTN node so mobile agents can travel from DTN node to DTN node carrying their application data. However, these platforms cannot be directly used for implementing DTNs.

Mobile agents are very useful when the studied scenario is geographically distributed, the environment is dynamic and interaction among the scenario software entities helps with problem solving. Mobile agents have the following characteristics as described in [99]:

- **Autonomy.** The assumption that, although we generally intend agents to act on our behalf, they nevertheless act without direct human or other intervention and have some kind of control over their internal state.

- **Reactive.** Capable of maintaining an ongoing interaction with the environment as well as to responding in a timely fashion to changes that may occur in it.

- **Pro-active.** Capable of taking the initiative; not driven solely by events, but capable of generating goals and acting rationally to achieve them.

- **Goal-Oriented.** An agent accepts high-level requests indicating what a human wants and is responsible for deciding how and where to satisfy the request.

- **Collaborative.** An agent does not blindly obey commands, but has the ability to modify requests, ask clarification questions, or even refuse to satisfy certain requests.

- **Mobile.** An agent is able to transport itself from one machine to another and across different system architectures and platforms.
- **Adaptation.** Agents may in some cases attempt to adapt themselves to better suit their new or changing environment or to deal with new or changing goals.

The key issue for using mobile agents among the different types of mobile code is to profit from the interdependency, meaning how these mobile codes autonomously behave. They are able to take decisions and reactively behave in terms of the context environment on behalf of the represented application. Cooperation among different agents belonging to the same application is also possible.

Mobile agents structure include an application code, which defines the behavior of the agent, the application data, which is the data carried by the software mobile agent and its itinerary, the node list the software mobile agents needs to visit. This list can be statically joined in the agent data or can be calculated by its code on every migration process.

None of the proposals in the DTN state of the art published studies solve the problem of the necessity of coexistence of different routing algorithms, as a general purpose solution. Classical DTN solutions listed in this section do not consider the strong dependency among routing algorithms and their applications. These strategies only take into account bundle layer information. Other layers, like the application layer containing some precious information, are not considered. Moreover, in these proposals, the criteria for bundle ordering or bundle dropping concerning the different applications do not change in each node. In the following chapters we will introduce a novel-general-purpose architecture which will satisfy precisely these limitations.

Part II

Proposal

"Copy Routing code (CP) bit. If set, this field indicates that the content of the code block data field should be copied to the absolute path indicated in the RIT path field."

"Wanderer, there is no road, the road is made by walking.", Proverbios y cantares. Campos de Castilla

Antonio Machado

"Packets cannot feel. They are created for the purpose of moving data from one system to another. However, it is clear that in specific situations some measure of emotion can be inferred or added.", RFC 5841

R. Hay and W. Turkal

3

DTN Architecture proposal

THE INTERNET is the evidence of how flexible and sturdy the TCP/IP network architecture has proved to be to create a global network of networks. Albeit there are many important issues about this architecture, there are also a good number or reasons to commend it. But there are some networks for which the TCP/IP is not an option because of, among other things, its relatively short timeouts or its routing algorithm. This is the case, for instance, when several devices are able to communicate to each other only once in a while, being unreachable most of the time. The nodes in these types of networks could use their peers to asynchronously forward information for them from node to node to any point of the network, even if this process took a long time waiting for the opportunity of transmitting the information. This is not a new idea at all, and there are many architectures in the literature providing similar solutions, usually under the labels of Challenged Networks, Opportunistic Networking or Delay and Disruption Tolerant Networking (DTN).

In these networks, a critical issue, if not the most important, to deal with is routing. Because there is not a fixed topography of the network, the decision making process to choose the next hop among the neighbours cannot rely on it.

Instead, routing has to be a dynamic function based on the local information acquired by the router in the very moment of the routing process. Influenced by the success of the Internet, full of general purpose mechanisms, one could think that a holistic "one-fits-all" routing algorithm would be useful here and establish a standard for opportunistic networks. Unfortunately, the approaches trying to do that are bound to fail. Such a standard, should it be, would lead to deadlocks similar to the ones faced in the current Internet (e.g. Multicast routing) because not all the applications have the same requirements and not all opportunistic networks are alike. Different applications may have different goals when it comes to routing the information such as minimising the latency time, minimising the variation in latency time, maximising the throughput, maximising the reliability or minimising the routing overhead. In fact, most of the successful solutions for doing the routing in these opportunistic networks fit just a particular scenario, normally consisting of a single application [30], and they set the trend. Some of these works claim to provide a generic framework for any type of scenario/application, but at the end of the day they fail to include new routing policies once the network has been deployed. On the other hand, it would be very convenient to let the applications decide the routing instead of trying to design a common algorithm including the associated fundamental properties for all scenarios and applications. This is definitely the way of optimizing the performance of this type of networks: there is no better source for routing information than the applications themselves.

In this chapter we introduce a general purpose architecture for DTN based upon the idea of letting the application, by means of its messages, decide the routing that will take place in every node. The keystone of this proposal is carrying the routing algorithm code along with every single message. The resulting DTN can be used by different applications, even if they were not foreseen before the deployment of the network. Thus, the DTN is no longer bound to specific applications and becomes a real open general purpose heterogeneous network as it has been on the Internet for connected networks. A network built following this architecture is flexible, open, and application designers can make the most appropriate use of it according to the specific requirements of the application. In short, the evident dependence between an application and its routing protocol is preserved while keeping other applications' interests.

This chapter presents a general architecture for opportunistic networks based on mobile code. A formal definition of all the actors involved in communication networks is presented. A generalisation of these communications is described

making our proposal a general case of communications information exchange. Several DTN issues such as routing, DTN congestion and error handling are discussed.

3.1 Motivation

We have seen in the previous chapter the existence of different applications which run on scenarios where there are intermittent connectivity, asymmetric bandwidths, long and variable latency or ambiguous mobility patterns. In these scenarios, the network is created when mobile and/or static nodes holding wireless devices intermittently connect. These network limitations are due to issues such as small wireless radio range, mobility of the nodes, energy constraints, network attacks or noise. Classic network protocols, such as TCP/IP, as explained in previous sections, do not solve these challenging issues. For these purposes, as described in Chapter 2, different proposals such as the Bundle Protocol [84] and Haggle [83] have been floated. These proposals evolve around the *store-carry-and-forward* paradigm: intermediate mobile or static nodes accept the custody application information until they are able to find new custodians to forward this information.

For every different application which is intended to be run on a DTN scenario, we get from the mentioned proposals, different interesting solutions which include various ways of solving routing problems. These proposals include replication based protocols, such as epidemic and PRoPHET [30], forwarding-based protocols, such as MEED [30] and LAROD [30] and a great many others as explained in Chapter 2. We believe that these proposals have been extremely useful to understand the complexity of the problematic issues around DTN networks. However, these different proposals are studied separately, defining a custom DTN network for every different scenario.

We propose a step further in DTN networks. We believe in the necessity of a common network in which different applications may coexist without leaving behind their optimal routing algorithms. Traditional bundle-based proposals [84] allow just one type of routing algorithms. When sharing a unique DTN infrastructure among several applications, a unique routing algorithm must be chosen. We believe this is not efficient and can be improved. Our proposal intends to break this tightly coupled relationship among DTN infrastructures and the applications.

Contrary to well-established protocols like TCP/IP, a unique routing algorithm for many applications has proven to work efficiently, and the Internet is the most compelling proof of it. However, there are different applications in the context of DTN networks, which work optimally just when using different routing algorithms. If we want to let these types of different applications share a unique network infrastructure we need to propose a new paradigm. This paradigm will be described in this chapter.

3.1.1 IP Multicast: a Case Example

Multicasting, in the context of TCP/IP networks and concretely in IPv4, is the sending of information over a TCP/IP internet to multiple destinations with a single datagram. Different applications such as software distribution, multimedia streaming, mobile TV, radio broadcasting and new media services, with different needs, take advantage of Multicast in order to spread application data all around the Internet.

Datagrams sent to Multicast groups have as their destination IP addresses an Internet address belonging to a special range of addresses called D type. Internet hosts subscribe or unsubscribe dynamically to these Multicast groups to receive or stop receiving application data from the application source.

It is the network infrastructure which is responsible of replicating the Multicast datagrams and intelligently routing these according to the topology of receivers interested in that information, that is, Internet hosts who are subscribed.

On one hand, hosts advertise to their local routers about their interest to belong to a given Multicast group, establishing the Multicast group membership.

On the other hand, several different protocols have been defined to communicate among local and remote Multicast routers in order to route Multicast traffic from the Multicast source application to the many Multicast clients. Examples on these protocols can be found in [79] and include protocols such as PIM Sparse Mode, PIM Dense Mode, PIM source-specific Multicast and Bidirectional PIM. These protocols are widely deployed all around the Internet.

Applications willing to use Multicast use the different network infrastructures using the very same protocols to route Multicast datagrams. Unfortunately, no general purpose Multicast routing algorithm covering every router on the Internet is available. This issue forces sender applications to choose from the different protocols in terms of the better adjustment of their needs and access

to the network infrastructure. For example, service providers offer multimedia services to their clients using their network infrastructure which employ optimal Multicast routing protocols chosen in terms of the type of multicasted applications, but do not work well for other types of applications. These networks remain isolated for other applications willing to Multicast since different PIM protocols do not understand each other.

The result, no global Multicast service is available on the Internet. Unfortunately, Multicast is not the only domain in which the coexistence of different protocols in not entirely solved. In the context of mobile ad-hoc networks (MANETs), different routing protocols have been proposed, but when it comes to support heterogeneous MANETs, no common routing paradigm copes with all the desired requirements. Our proposal in this chapter intends to avoid precisely these types of failures in Delay and Disruption Tolerant Networking scenarios.

3.2 Scenario Description

DTN research, as described in Chapter 2, tries to provide solutions to problems where network partitions are created due to intermittent connectivity by using the already-described *store-carry-and-forward* switching. We propose in this study an enhanced version of this paradigm based on mobile code. In this section we firstly introduce the different actors and functions involved in DTN scenarios. Following this, a six-phase general case of protocol is described.

Application list. We define *applist* as a non empty set $\{\text{app}_1, \text{app}_2, ..., \text{app}_n\}$ as the list of n applications present in a network scenario. Applications mainly create messages which are sent over the network from a source to a destination.

Custodian List. Let $\{c_1, c_2, ...c_c\}$ be the list of c nodes capable of temporally store application messages, carry them and routing them to some other custodian nodes.

Application messages. Let $\{mess_{app_i,1}, mess_{app_i,2}, ..., mess_{app_i,m}, \}$ be the list of m messages an application app_i sends over the network. Let $messages_c$, be the list of messages a custodian node c has at a given time.

Custodian Routing Information Tree (RIT). For every custodian node c_c, routing information may be stored locally in the custodian nodes. A RIT may be defined as a root node, RIT_{root} or $t+1$ number of RIT trees $\{RIT_{local}, RIT_{app_1}, RIT_{app_2}, ..., RIT_{app_t}\}$ where t is the number of the different applications present in the network. These trees contain other RIT trees or values for routing purposes.

Custodian Buffer Space. The amount of space a custodian node c_i has available for application messages and routing information is defined as $buff(c_i)$.

Custodian movement model. Custodian nodes follow movement models m_{c_j} which affect important network parameters such as connection window time and encounter statistical distribution.

Application Custodian Code. Let $code_{app,c_j}$ be a application code an application *app* intends to execute on a custodian node c_j.

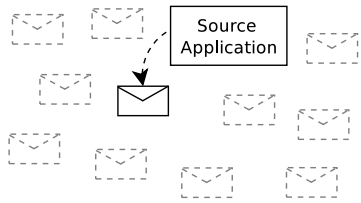
Neighbour List. We define $neighbourList(c_j, t)$, as the list of p neighbours $\{n_1, n_2, ...n_p\}$ a custodian node c has at a given time t.

Routing function. Given an application message $mess_{app,i}$, belonging to application *app*, a custodian node c_j, and its local RIT tree RIT_{c_j}, let $f_{routing}(mess_{app,i}, neighbourList(c_j, t), RIT_{c_j})$ be the function which returns a subset of the neighbours of the custodian node c_j which should custody the application message.

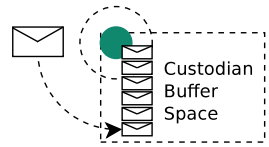
In general, network protocols follow three traditional independent phases: Data creation phase, routing and delivery phase. Besides the above-mentioned *store-carry-and-forward* paradigm, we define the following communication paradigm phases:

Data creation phase. We define data creation for application *app* as a process in which application *app* creates m application messages $mess_{app,i}$, i in $[0..m]$. Application messages may be created as the result of application programs or application custodian codes ($code_{app,c_j}$). The format of data messages will be described in Section 3.3.

Store phase. Messages are stored in a custodian node c_j using the $buff(c_j)$. Since this space is limited, the routing problem can be considered as an optimal resource allocation challenge.



(a) **Application message creation phase**. In this phase, application *app* creates $mess_{app,i}$.



(b) **Store phase**. Message $mess_{app,i}$ is stored at $buffer(custodian_c)$ in custodian $custodian_c$.

Carry phase. Messages are carried from one place to another. Messages perform **active carries**, that is, carrying application *app* information from custodian node to custodian node in terms of the function $f_{routing}(mess_{app,i}, neighbourList(c_j, t), RIT_{c_j})$. Instead, a custodian node if in motion, performs a **passive carry** when moving. In this case, there is no software entity performing the transport itself. Instead, there is a physical movement of the custodian node $m(c_c)$ who performs the **carry** action.

Process phase. A custodian node c_j may execute $code_{app_i,c_j}$ on behalf of the application *app*. For these purposes, custodian need to have execution environments. This phase is equivalent to the *process* phase in the *store-carry-process-and-forward* paradigm that will be explained in Chapter 6.

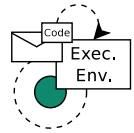
Forwarding phase. Given custodian node c_j and application information message $mess_{app,i}$ belonging to application *app*. This phase is equivalent to the forward phase in the already defined *store-carry-and-forward* paradigm. Message $mess_{app,i}$ is propagated in the network following the following expression. Note that if $forwardToList = \emptyset$ it means that the message is discarded.

$$forwardToList = f_{routing}(mess_{app_i}, neighbourList(c_j, t), RIT_{c_j}), \text{ where}$$
$$forwardToList \subseteq neighbourList(c_j, t) \cup c_j$$

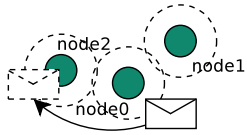
Delivery phase. Once a message $mess_{app_a,i}$ has arrived to its destination it is passed to the destination application and removed from the last custodian

buffer ($buff(c_{destination})$). Other copies of message $mess_{app_a,i}$ may be present in the network in other custodian nodes.

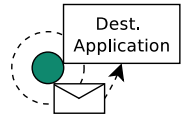
(c) **Carry phase**. The Message follows the custodian $movement(custodian_c)$ movement model.



(d) **Process phase**. The custodian node c_j executes local code $code_{app_a,c_j}$ on behalf of application app_a.



(e) **Routing phase**. The message is replicated following its routing algorithm, which in this example returns *node2*.



(f) **Delivery phase**. Message $mess_{app_a,i}$ is freed from last custodian buffer ($buff(c_j)$) and delivered to the application layer on custodian c_j.

We propose the idea of creating a network to allow different applications coexist without leaving behind their optimal routing algorithms. Our proposal is leaving this routing decision to the very same application. This will let the different applications determine which routing algorithm to apply in every DTN hop. In order to achieve this flexibility two options must be studied.

Firstly, a deployment of all of the possible routing protocols $f_{routing}(mess_{app_a,i}, neighbourList(c_j, t), RIT_{c_j})$ in the different DTN custodian nodes ($\{c_1, c_2, ..., c_m\}$). This option presents some immediate disadvantages. Routing algorithm deployment is expensive in challenged networks and it is not

guaranteed that this deployment of all the necessary $f_{routing}$ functions from the different applications will arrive at every custodian node. Furthermore, newcomer applications will need to make additional routing deployments.

Secondly, we could let the very same application message $mess$ carry along with itself the routing algorithm $f_{routing}(mess_{app_a,i}, neighbourList(c_j, t), RIT_{c_j})$ and/or the application custodian code, $code_{app_a,c_j}$. Theses algorithms must be able to be run in any custodian node. Mobile code is a good candidate to implement a data-driven approach to DTN networks.

In some scenarios with limited storage space having replicated the same routing algorithm code on every message belonging to the same application, could be inefficient. Every time the active message is forwarded, besides message data, the routing code must be sent to the destination platform. These static codes can be cached on the different node execution environments, using as described in [33], a global cache service to efficiently deal with the distribution of agent codes. Caching these codes improves and speeds up message forwarding.

3.3 Active Messages

Messages containing application data, routing algorithms $f_{routing}(mess_{app_a,i}, neighbourList(c_j, t), RIT_{c_j})$ and application codes $code_{app_a,c_j}$ are called active messages. In Figure 3.1 the four fields that define an active message are depicted. These fields are:

- **Delivery Information.** In this field the source and destination addresses, active message creation timestamp, and static lifetime is defined.
- **Application Code.** The application code $code_{app,c_j}$ creates and manipulates application data. This data may be created beyond the source custodian node. Data belonging to an active message is reviewed every time the application code is executed on a custodian node.
- **Routing Code.** Routing algorithm code $frouting(app_a)$ chooses from the available neighbours the list of the potential information custodian nodes. The routing code will be responsible for defining the path the application data will take.
- **Data payload.** Application data created by the source application.

We propose a simple paradigm, depicted in Figure 3.2, in which the application information is carried by the active messages. Active messages can be seen therefore as a transmission medium, i.e., each active message is functionally analogous, conceptually speaking, to a carrier pigeon, as in [96]. DTN custodian nodes must include an execution environment which implements the DTN layer that handles delays and disruptions. The application code creates or modifies application data which is encapsulated inside the active messages. Active messages stay in the custodian nodes which accept their custody, until they are able to be forwarded to another one or they arrive at their destination node. Custodian nodes may employ different convergence layers (as defined in Chapter 2), the interfaces between the common Bundle Protocol and the specific inter network protocol suites.

Figure 3.1: Active message fields: control information, routing algorithm and data payload.

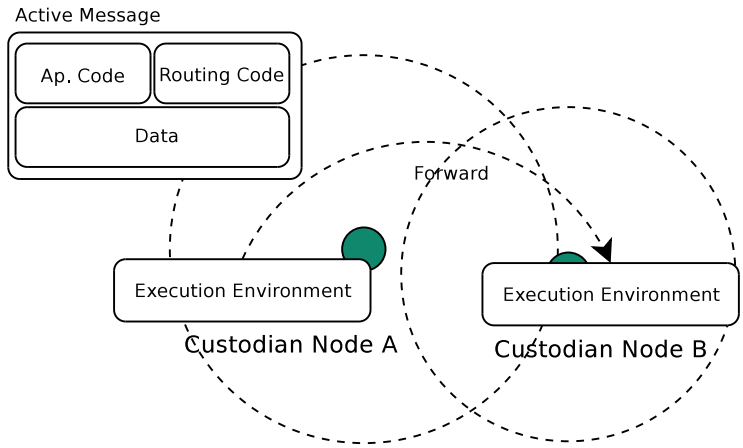


Figure 3.2: Active Message forwarding. Custodian Node A forwards an active message containing some data, its routing code and an application code to Custodian Node B.

3.4 General Protocol Architecture

Layering protocols is an excellent way to simplify networking designs by the means of separating the latter into functional layers, and defining protocols to delegate each layer's task. In the context of network architectures based on layers, every layer receives from the previous layer its *protocol data unit*. From the moment of this reception, this *protocol data unit becomes* the *service data unit* of the new layer, which upon adding a header from the new layer, a protocol data unit of the new layer is obtained. Each layer interacts, mainly with its neighbouring layers, across the interfaces between them. This mechanism is called encapsulation and guarantees the layer principle in which different layers are independent from one another. Traditional de facto standard protocols like TCP/IP break this layer principle for practical reasons, letting some information from one layer be employed by its following layer.

On the other hand, most of the network architectures propose uniqueness when it comes to choosing their different protocols implemented by the different layers. Information travelling on networks defined by these architectures permits different protocols on the different layers, but on every hop, these protocols are limited to the ones chosen by the layer's code. For example, in the context of IP communications, we understand that a datagram is routed on the basis of the information gathered from a certain internal router protocol, such as RIP or OSPF, but we would not see it as feasible that the datagram could choose among these two mechanisms in order to be routed.

Layered protocols do not only imply just a separation of responsibilities. Layers are placed one after the other, and define in which order things should be done. In TCP/IP for example, routing algorithms are executed before the translation of logical addresses into physical addresses while sending a datagram.

We propose simple DTN architecture which is based on layers but in a very particular way. This architecture is fully compatible with the DTN proposals defined by the Delay-Tolerant Networking Architecture [18] as it will be described in Chapter 4. A layer based model, according to the definition explained in this section, implies delegation of tasks, encapsulation, uniqueness, and should define a sequence of actions.

In Figure 3.3 our DTN architecture is depicted. An application layer represent the different application which may employ the network. The DTN layer handles problems such as addressing, routing, reliability and custody transfer,

congestion and security in a very similar way as in the Delay-Tolerant Networking Architecture with a subtle difference. The DTN layer may be different for every application. Issues such as routing, reliability and custody transfer, congestion and flow control, security may be treated differently depending on the application.

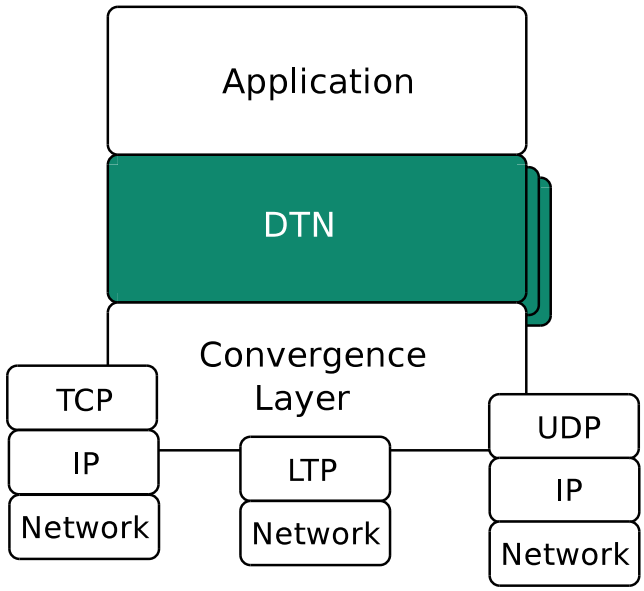


Figure 3.3: General network DTN architecture. The different levels in the DTN layer represents the possibility of employing different routing protocols and different DTN behaviors. The different convergence layers such as Licklider Transmission Protocol (LTP)[20], provide communication among intermediate nodes when they meet in an opportunistic manner.

3.5 General System Architecture

In this section, the different system architecture elements are described.

- **RIT Manager.** Routing information is classified into ontologies, which represent the different domains from the different applications which may

coexist in a network. The RIT manager is responsible for the access and storage of this information, which can be created when nodes meet and exchange application information, or with the output from active messages execution codes. The very same active messages may classify, modify, access or remove information from the *Routing Information Tree* contacting the RIT manager. Local nodes limit, remove, order, index or purge the content of the RIT depending on their resource constraints. In Chapter 4, the RIT manager will be explained in detail.

- **Local Routing Information Generator.** Local routing information may be created by the local nodes for future use of routing algorithms. Algorithms like PRoPHET [30], need local routing information which should be updated every certain time and every time a node is contacted. The information created by the *Local Routing Information Generator*, is stored in the *RIT* following a certain and well defined order in order to allow active messages to find the necessary information for its routing decisions. In Chapter 4, the way this information is created and stored will be explained in detail.

- **Network Abstraction.** A module called *Network Abstraction* is defined to permit local nodes interact with different heterogeneous networks. Basic primitives, *send*, *receive* and *neighbour discovery* are available to the nodes. The primitives *send* and *receive* allow transmitting and receiving bundles of information among nodes which can be of three different types:

  - *Active messages.* As introduced in Section 3.3, These are messages containing application data. The content of these messages include application data, application code and routing code.
  - *Beacon messages.* Beacons are sent at certain intervals of time to allow neighbours to be discovered. These messages contain a node identification using Universal Resource Identifiers (URI's) as defined in [9]. Along with the node identifier, basic information that describes the node may also be included in the beacon messages. Which information from the different fields in the RIT should be announced is not a trivial issue. In order to be flexible enough, we allow the very same application messages flag the desired RIT fields among the RIT application branch to be announced.

- **Scheduling and Forwarding Manager.** Two queues for incoming and outgoing active messages are managed by the *Scheduling and Forwarding Manager*. A prioritised scheduler manages which active messages should be forwarded and accepted first and for discarding purposes. The way these priorities are broadcasted will be discussed in Chapter 4. For ordering or discarding active messages, we will also consider information from other layers, like the application layer. Other proposals for ordering DTN just take into account information from the DTN layer. We believe information from the DTN layer is important, but information from the application layer is crucial to distinguish which are important messages and which are not. More information on how messages are scheduled can be found in Chapter 4.

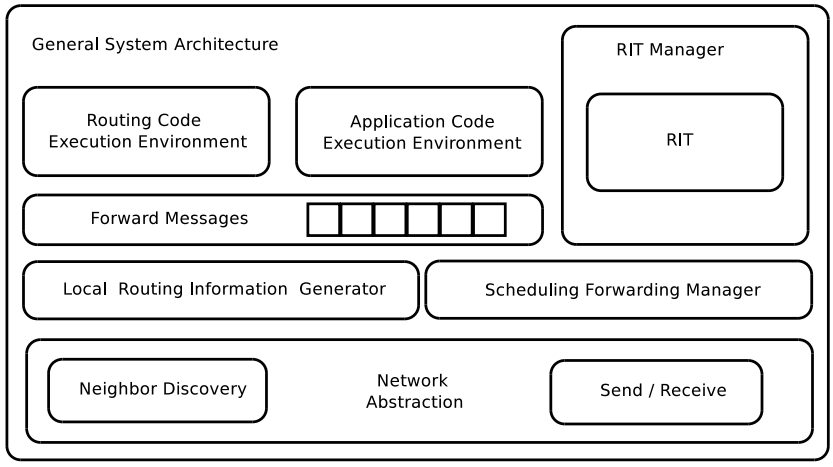


Figure 3.4: The elements that form the DTN proposed system architecture.

- **Pool of Forwarding Messages:** Active messages are accepted by the local nodes in the *Pool of Forwarding Messages* following the Delay and Disruption paradigm *store-carry-and-forward*. Storage reserved for these purposes is a key issue in opportunistic networks. Routing algorithms must prevent from making epidemic decisions which may flood buffers containing the pool of messages.

- **Routing and Application Code Execution Environment:** As already explained, active messages contain code for routing and application behavior purposes. In order to execute this code, an execution environment must be included on every DTN node of the network. In Chapter 4, the code execution environment is explained in detail. Two examples of execution environments to be used in DTN scenarios are presented.

3.6 Security Considerations

We are aware that security is a very important issue. While designing our architecture security has been fully taken into account. Even if in this thesis is not directly covered, in this section we want to enumerate the list of security considerations which should be studied. These mechanisms must guarantee the following properties:

Data confidentiality. The confidentiality of the data provided by the application sources and sent to the destination nodes must be warrantied, otherwise an attacker could perform some malevolent actions in order to eavesdrop on this information.

Data authenticity. Authenticity is vital to avoid any sending or injection of false information destined for a DTN node. It is also important to remark that this authenticity should not be linked to the identity of the node that generated the information.

Data integrity. The integrity is crucial to permit the detection of a state where a compromised node has modified the data which comes from the application sources, or the data which it forwards to another node in a routing process.

Since the active messages are one of the pillars of our proposal, and since their code can influence the movement and the behaviour of a node, it is important to protect it to avoid undesirable circumstances. In this case, we consider the following security properties as the most appropriate.

Code confidentiality. The confidentiality of the code is a desirable property to avoid outside attacks, which can include eavesdropping on the radio channel, the goal of which is to profile the kind of information that an

application is gathering, and the procedure used to obtain it. Although this requirement can be desirable in some cases, this is not a mandatory property and it depends on every particular application.

Code integrity. The authenticity of the code of an active message must be warrantied in order to avoid an illegitimate active message code being executed in a node. Otherwise, an insider or outside attack could inject a malicious active message with the purpose of compromising the security of the infrastructure.

Node identification. Since the custody of a message is delegated to DTN nodes, the correct identification of a node is a very important issue. A security mechanism should be defined to avoid an illegitimate DTN node to impersonate a legitimate one.

Key management. In the case symmetric or asymmetric keys are employed in the security mechanisms proposed in this section, a key management protocols should be defined. Issues like revocations lists deployment are complicate tasks in intermittently connected scenarios.

One of the most important aspects of our proposal is the use of the active messages as a way of routing the code that must be used to take the measurements. This can be done by using active messages and by sharing contextual routing information between the nodes. Thus, it seems evident that security must be present in the routing procedures. We impose the following minimum requirements to guarantee the security of the routing:

Routing data authenticity. The authenticity of the contextual routing data shared by nodes must be preserved in order to avoid that any outside attacker injecting incorrect information. This property can also be seen as a way to deter any potential outside attacker, since they cannot provide this property. It is clear that this requirement does not allow us to detect any malevolent data injected by an inside attacker.

Routing data integrity. Integrity must be preserved with the aim of avoiding an attacker modifying the routing data, as this change is not detected by its receiver. Again, this property cannot be preserved in the case of an inside attack, but it is also one way to dissuade any potential outside attackers.

Active messages forwarding non-repudiability. The non-repudiability property in the process of an active message forwarding must be warrantied from the source and the destination perspective. This is usually known as strong fairness, and allow us to detect when a compromised node has completed a malicious action from the perspective of the routing of the active messages. For this purpose, all the non-repudiability evidence must be correlated to detect when an inside attack has performed a malicious behaviour. This correlation must be done in places considered points of trust.

"Most of life is offline, and I think it always will be; eating and aching and sleeping and loving happen in the body. But it's not impossible to imagine losing my appetite for those things; they aren't always easy, and they take so much time. In twenty years I'd be interviewing air and water and heat just to remember they mattered.", It Chooses You

Miranda July

"Life is not like water. Things in life don't necessarily flow over the shortest possible route.", 1Q84

Haruki Murakami

4

Architecture Reifications

FOLLOWING THE ARCHITECTURE proposed in Chapter 3, in this chapter the concept of active messages will be reified. The objective of this chapter is to make the abstract concept of active message more concrete in order to be able to implement it and apply it in concrete scenarios. These concretizations are called all over the thesis *reifications*, which comes from the Latin words *res* (thing) and *facere* (to make). In this chapter we propose to treat the abstract *Active Message* as if it had concrete or material existence.

The first reification, described in Section 4.1, is based on mobile agents. The Jade [6] architecture with a mobility add-on called IPMS is described in [32] is presented. The different changes applied to this software to allow DTN applications to use it will be described. Additionally, a dynamic prioritised queue for messages will be described. In Chapter 5, an example of this reification will be described.

The second reification is based on a multi agent platform, Mobile-C [26], and will be described in Section 4.2. An example of this reification in the context of Grid Computing [11, 46] will be described.

A third reification is presented in Section 4.3 based on the Bundle Protocol

[84]. In Chapter 7, an example of this reification is presented in the context of emergency scenarios.

4.1 A Mobile Agent-Based Architecture Reification

In this section a first implementation for the architecture proposed in Chapter 3 is described.

The first attempt for an implementation of the active messages and the architecture presented in Chapter 3 was a mobile agent-based approach. Mobile agents are software entities capable of migrating with their code, along with their data and state, from machine to machine and resume their execution on the destination machine. As described in Chapter 2, mobile agents are very useful when the studied scenario is geographically distributed, the environment is dynamic and interaction among the scenario software entities helps with problem solving. We propose letting the mobile agents carry the application data and migrate from DTN node to DTN node until the final destination is reached. The entities on which mobile agents run are usually called platforms, agencies or sometimes just execution environments and should be installed on every DTN node.

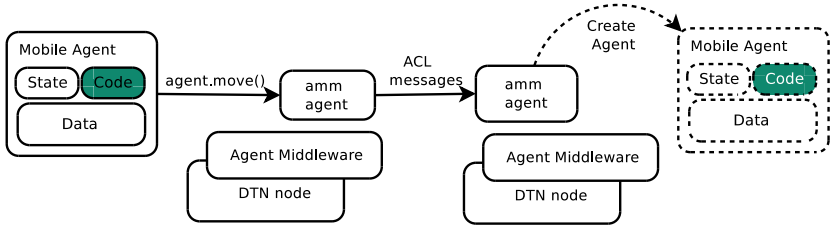


Figure 4.1: Mobile agent migration. A mobile agent migrates from one mobile agent middleware with its state, code and data, to another mobile agent middleware. Messages are designed to contain all data necessary for agent communication.

A Java implementation based on Jade [6] with a mobility add-on called IPMS described in [32] was a first attempt to provide a real practice of the proposed

DTN model. IPMS is a mobile agent environment which provides platform-to-platform mobility for Jade agents. The benefit of using this environment is that there is no centralised point which manages every agent migration. DTN nodes must be independent enough to ensure flexibility; so this approach suits the architecture introduced in Chapter 3 well.

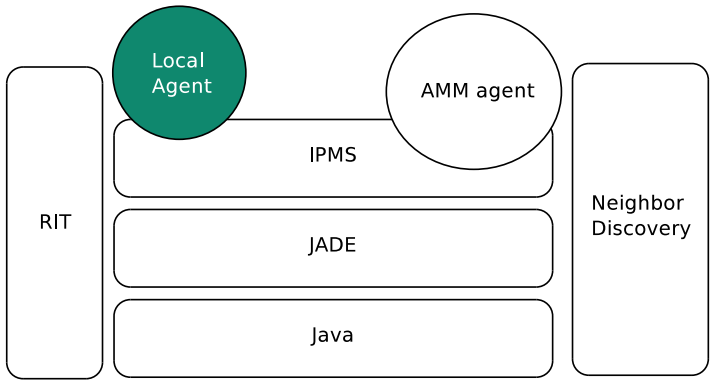


Figure 4.2: A mobile agent Java Jade-based architecture. The IPMS service allows the inter-mobility of mobile agents. The Agent Mobility Manage handles mobile agent migrations.

The IPMS add-on manages a list of agents which are ready to migrate. This list follows a FIFO criteria on agent releasing. We have modified the IPMS add-on to include a priority value in the agent list. In order to achieve this, one of the behaviours belonging to the agent in charge of the migrations is changed. This priority can be either specified by the mobile agent or calculated by the platform before the agent is pushed into the agent list. The agent's variables are the input data to calculate the agent's priority plus the local context. The algorithm can be updated instead by the agents themselves at any given point and it can be different from one DTN node to another.

In every custodian node a structure is added to calculate agents priorities. This structure is depicted in Figure 4.3. The reader can see this structure as a routing information tree in which agents themselves are able to read and write in order to keep it updated. The list of expressions to calculate the agent's priorities differs from application domain to application domain. As depicted in Figure 4.3, we separate the different *ontologies*, that is, application domains,

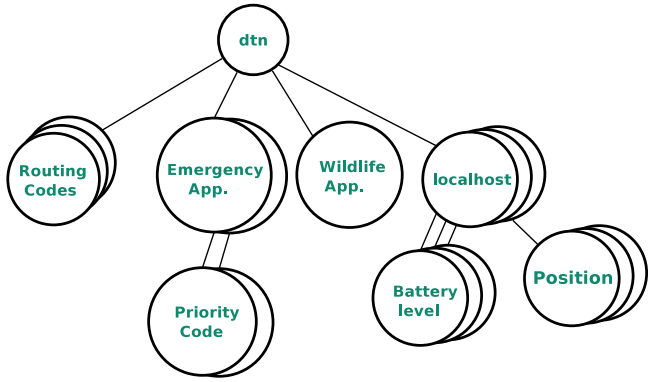


Figure 4.3: Example of a Routing Information Tree ontology structure. The information needed to make routing decisions is ordered in a tree-like structure. Different applications read and write from different branches of the tree. Local information is store as well under the *local* branch.

that can coexist. Expressions belonging to the emergency scenarios agents are under a different branch from those belonging to air field scenarios. Agents easily access this shared structure by following the *ontology* tree, in order to retrieve the corresponding expression. As presented in Chapter 3, this structure is called Routing Information Tree (RIT).

The *agent mobility manager*, that is, the agent which handles agents platform migration and immigration (amm agent), creates the list of agents prepared to migrate and their corresponding priority. These priorities are calculated using the *RIT* structure and the mobile agent's matching variables which can be part of *RIT* expressions under their ontology. For example, in the case of emergency scenarios proposed in this section, as seen in Figure 4.3, priorities from mobile agents belonging to *Emergency scenarios* ontology are calculated in terms of the victim state. Nevertheless, this function, as explained in Chapter 3, can vary on time as it is updated by the mobile agents themselves.

The algorithm to describe how the application code behaves is described in Algorithm 1. In Algorithm 2, the code executed by the *amm agent* is described.

In Chapter 5, an application example of this Active Message reification will be explained in detail.

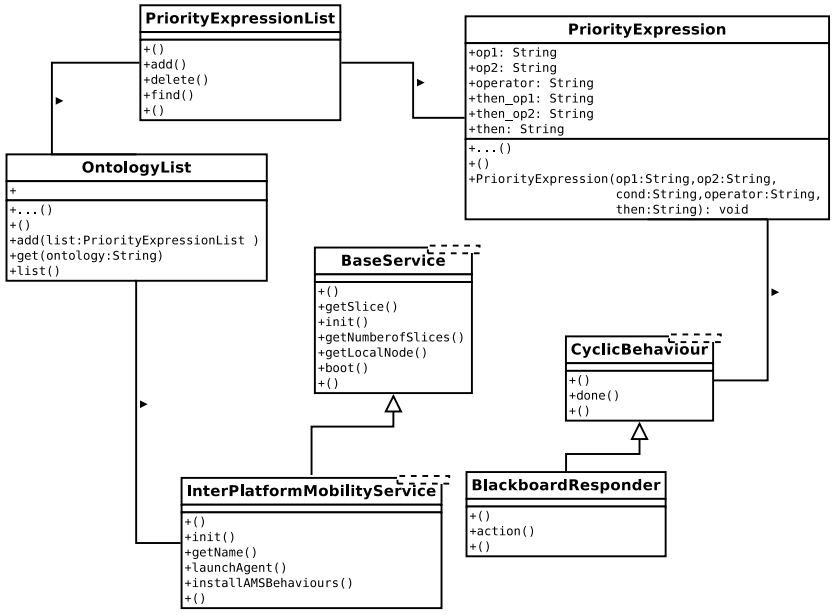


Figure 4.4: JADE implementation. The classes which are modified in the IPM-S/Jade platform mobile agent implementation are depicted. This includes a RIT implementation and a prioritised scheduler.

4.2 Using Mobile Agents as Data and Code Containers

In order to improve the performance of our proposal in certain scenarios, a second implementation of the architecture proposed in Chapter 3 has been conducted. For these purposes, a generic and multi-platform solution which provides with mobile code and data transport has been searched. Among the different available mobile code execution environment solutions, we have decided to use Mobile-C, a Foundation for Intelligent Physical Agents (FIPA) [6] complaint multi-agent platform for mobile C/C++ codes. The benefits in comparison to other mobile code execution environment platforms like Jade (see Section 4.1) based on JDK1.4 are mainly achieved through the reduction in memory consumption, moreover, in general terms, the impact on system resources is small.

Algorithm 1: Application agent code. This code needs to be included in the mobile agent to perform the migration action.

```
Behaviour: AgentMigration
1: function ACTION()
2:     //Agent migrates to an available platform
3:     myAgent.doMove(nextHopApplicationLayer);
4: end function
```

Mobile-C uses an embeddable C/C++ interpreter called Ch described in [26], to support the execution of C/C++ mobile codes. This interpreter should be included in every platform running on every DTN node. Besides, a library is provided with an easy-to-program API which facilitates the development of multi-agent systems making it very easy to interface with a large variety of hardware devices.

Once application data is created, active messages travel from DTN node to DTN node, carrying data heading the application data destination. Platforms custody active messages until another DTN node becomes available for forwarding. The very same active message chooses its DTN node path when a new platform becomes available by executing the *MC_AddActiveMessage()* method, as described in Algorithm 2.

The main difference between the previous reification based on mobile agents and this one lies in the use of the mobile agents themselves. While the previous approach the mobile agent may have different behaviours which implement the different tasks an application delegates to the agent, in this new approach the mobile agents are employed just as mobile and data containers. The functionality is reduced to specific functions which are launched by the local platform code.

We have modified the Mobile-C platform code to allow having different well-known methods which can be invoked for different purposes such as the routing decision, task processing, etc. For example, in Algorithm 3 an epidemic version of the route function is described. Additionally, this function is invoked by the local platform to let the active message choose among the available platforms, as seen in Algorithm 4.

In order to centralise the structure of Mobile-C and make it similar to the one used in JADE, agencies can be started with or without the support to

Algorithm 2: Active message code. Active message is prepared, started and processed.

```
activemessage = MC_ComposeActiveMessageFromFile(
        "activemessage1", /* Name */
        "localhost:5050", /* Home */
        "IEL", /* Owner */
        "activemessagecode.c", /* Filename */
        NULL, /* NULL for no return */
        "nextHopApplicationLayer:5051", /* Server */
        0 ); /* no persistence. */
        /* Add the routing code to the active message */
        MC_ActiveMessageAddRoutingCodeFromFile(activemessage,
"routing_code.c");
/* Add the active message to the platform to start it */
MC_AddActiveMessage(platform, activemessage);
MC_MainLoop(agency);
MC_End(agency);
```

Algorithm 3: Active message routing code. Epidemic example. The message is replicated on every neighbour node found.

```
Behaviour: neighbours
Output: ∅
function ROUTE()
    for node in neighbours do
        mc_SetActiveMessageNextHop(mc_current_activemessage,
        "node:5050");
    end for
end function
```

Neighbour Discovery. The idea is to start one principal agency with support to Neighbour Discovery and make use of the secondary agencies without support to Neighbour Discovery only to launch the agents. Agent's routing code should be programmed in order to make first a jump to the principal agency, and then, continue its routing (or execution) as normally. Enable or disable that support to Neighbour Discovery involves using the *enable discovery* flag of the

Algorithm 4: Local platform code. The *route* function is called by the local platform for every active message.

```
1: // During the creation of the migration message
2: if activemessage-> datastate->is_routable == 1 then
3:     destination=Ch_CallFuncByName(
4: *activemessage->activemessage_interp, "route", NULL);
5: end if
```

MCAgencyOptions_t struct. By default, that flags has value 0 (disable).

The platform keeps the queue of the mobile agents which are waiting to migrate. As seen in line 2 in Algorithm 5, the *amm agent* checks whether the first agent in this list queue implements a concrete class called *RoutableAgent*. If it does, this means that the platform is able to execute a given method called *nextHop*, formally represented as $f_{routing}(mess_{app,i}, neighbourList(c_j, t), RIT_{c_j})$. It receives as arguments the list of neighbours, its application layer next hop and the shared infrastructure or *Routing Information Tree* and returns the platform the mobile agent would like to migrate to. While executing this method the agent is not *woken up*. A mobile agent could eventually return none, then as seen in line 9, the mobile agent is punished as described before. However, not all the scenarios are appropriate for punishing mobile agents. This decision will depend on the different applications. By using a configurable *punish criteria* in the platform code we help different application scenarios to coexist.

As seen in Figure 4.5, we have to differentiate among two different levels while defining the *nextHop*. From the application layer point of view, there is just a *NextHop Application Layer*, that is, the final destination for the mobile agent. From the DTN/mobile agent layer point of view, there are several neighbours running mobile agent platforms, which are not the final destination for the application layer. The mobile agent *nextHop* method chooses among all of them to define which will be its *Neighbour Immediate Nexthop*. This *Neighbour Immediate Nexthop* is an agent platform in which the mobile agent will not execute any code but its *nextHop* method, jumping from *Neighbour Immediate Nexthop* to *Neighbour Immediate Nexthop* until it reaches its *NextHop Application Layer*.

In Chapter 6, an application example of this Active Message reification will be explained in detail.

Algorithm 5: The mobile agent mobility manager code. In this code the agents are selected in terms of their priorities and forwarded in terms of their routing codes.

```
Behaviour: AgentRouting
 1: function ACTION()
 2:     //See if agent implements the RoutableAgent agent
 3:     if firstagent implements RoutableAgent then
 4:         //amm selects best agent in terms of priority
 5:         nexthop = firstagent.nextHop(neighbours,
 6:         nextHopApplicationLayer,RIT);
 7:     else
 8:         if nextHopApplicationLayer in neighbours then
 9:             firstagent.doMove(nextHopApplicationLayer)
10:         else
11:             if nextHopApplicationLayer == none then
12:                 decrease_priority(firstagent)
13:             else
14:                 firstagent.doMove(neighbours.any())
15:             end if
16:         end if
17:     end if
18: end function
```

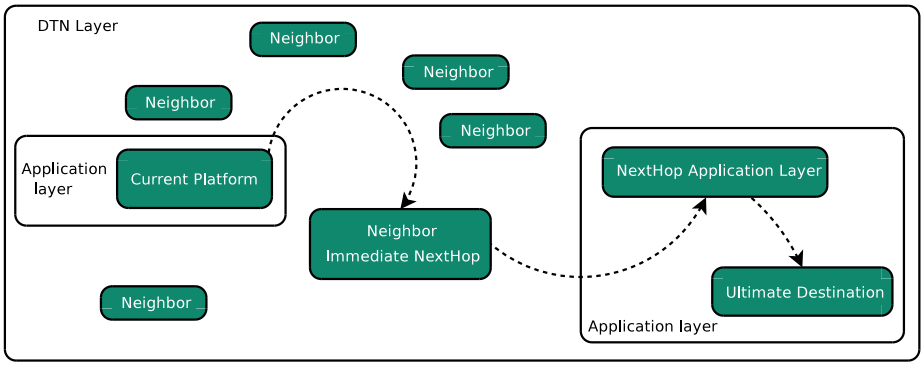


Figure 4.5: A mobile agent differentiates among two migration levels: *NextHop Application Layer* and *Neighbour Immediate Nexthop*.

4.3 Bundle Code Blocks Reification

In this section we integrate our DTN architecture proposal described in Chapter 3 with the DTNRG DTN architecture [18]. Various bundle extensions to allow bundles to carry mobile code for routing improvement purposes are presented. Our proposal is based on the novel idea of moving the routing algorithm from the host to the bundle. We introduce five types of code blocks that may be optionally included in a bundle.

For information storing purposes, a local custodian information structure in which the following-described bundle codes may read from or write to is defined for the common use of the extensions. This information is ordered following a tree structure similar to the Standard Management Information Base (MIB) [71], but in a more simple way. DTN information is written down under the *"/dtn/"* path and every node of the tree is separated by the *"/"* symbol. For example in *"/dtn/local/position"*, the current position of the custodian node may be found. Relatives paths, such as *"./local/position"* are also valid paths. Bundle routing algorithms may use basic primitives such as *get*, *put* and *search* to access the local information data structure.

Firstly, we propose a bundle extension to allow the bundle to carry the routing code which selects among the local custodian node neighbours the ones where the bundle should be forwarded to. This code will be referred to as **Bundle Routing Code**. Traditional DTN implementations keep this code

local within the custodian node, as in the Bundle Protocol, or at least, allow the DTN information to choose among a limited list of routing codes, as in Haggle [83]. In our proposal, the bundle itself is able to carry its own routing code, thus providing routing flexibility.

Secondly, some bundle routing codes need special code to be regularly executed on every custodian node in order to create or update certain information which will be used in the future by the very same bundle routing code. For example, probabilistic routing algorithms like PRoPHET [30], execute code on every custodian node in order to update the probability of finding a certain custodian node every time this node is contacted. Besides, these probabilities are automatically decreased based on a preset utility reduction method. This code will be referenced to as **Custodian Routing Code**.

Additionally, bundles may leave information stored in the custodian node RIT so future bundles belonging to the same application visiting the same custodian nodes may employ this information for routing decision purposes. The code which updates this guiding information, may travel with the bundle itself. This allows the applications to have a general-purpose way of communicating among the different bundles belonging to the same application. This code will be referenced to as **RIT Update Bundle Code**.

Bundles may carry another code to update the way the bundles are prioritised. Bundle prioritization, besides being something innovative, is a very useful way of avoiding important bundles from being blocked by other less important bundles. This may be very useful when bundles belonging to an application flow have different priorities. However, the criteria for bundle prioritization could eventually change and using an extension which carries these criteria may be very useful for that purpose. This code will be referred to in the following sections as **Priority Bundle Code**.

Finally, a lifetime control code may be carried by the bundle in order to improve congestion. Bundles, like IP datagrams, include a *time-to-live* field to control the time at which the bundle's payload will no longer be useful. In the DTN architecture definition (see [18]), it is mentioned the necessity of a way to express the useful lifetime of data to allow the network to better deliver data in serving the needs of applications. Beyond IP's *time-to-live* and DTN bundle lifetime field, we propose using complex code expressions based on the local context or application wills, to be carried by the bundle for the same purposes. This code will be referred to as **Bundle Lifetime Control Code** in the following sections.

BlockType	Code
BundleRoutingBlock (BRCB)	0x10
CustodianRoutingBlock (CRCB)	0x11
BundleUpdateBlock (BUB)	0x12
BundleControlBlock (BCB)	0x13
ApplicationPriorityBlock (APB)	0x14

Table 4.1: Blocks Code Type Codes allocated in this proposal.

In the sections that follow, several bundle code extension blocks are defined to allow bundles to carry the above-mentioned codes.

4.3.1 Common Extension Fields

Every bundle code extension has a group of common fields which will now be described. This fields are depicted in Figure 4.6.

Block-type code (one byte) This proposal uses two code-points from the existing *Bundle Block Types* registry defined in [7]. Different code blocks will have different block-type codes, starting from the first available according to [7]. Our proposal allocates block type codepoints from 0x10 to 0x14.

Block processing control flags (non-fixed length field). Encoded in Self-Delimiting Numeric Values (SDNV) format as defined in [84], this field follows the bundle definition described in the Bundle Protocol. These bits are defined as the following:

- **Bit 0.** The block must be replicated in every fragment. Fragments will be treated the same way as the original bundle.
- **Bit 1.** Transmit status report if the block can't be processed. Applications may be informed if the code blocks are not processed.
- **Bit 2.** Delete the bundle if the block cannot be processed. Code block extensions are added to help the bundle to be routed. If the code cannot be processed the bundle agent (an agent in charge of handling the bundles) will delete this bundle if this bit is set.
- **Bit 3.** Last block. This bit is set if the extension block is the last block of the bundle.

- **Bit 4.** Discard the block if it cannot be processed.
- **Bit 5.** The block was forwarded without being processed. The block can be marked as not processed if the bundle agent is not able to process it.
- **Bit 6.** The block contains an EID-reference field. The bundle extension may include a reference to an EID contained in the bundle main block.

EID-references This optional composite field contains references to endpoint identifiers (EIDs). The 6th bit from the block processing control flags, the "Block contains an EID-reference field" should be set. If no EID-references are present, this field should remain empty. EID-references allow blocked code extensions to be ignored or employed only in certain custodian nodes.

Block data length Expressed in SDNV format as defined in the Bundle Protocol for every block except the primary bundle block.

Code Block-type-specific data fields include:

Code type. Encoded in SDNV format, this field is used to describe the type of code included in the extension. In this code we may differentiate among the possible code types which include the different executable formats and linkables executable formats, as well as the different bytecode and source code for the different programming languages. An example of the code for this field could be Java bytecode coded as *100*.

Version. Encoded in SDNV format this field is used to describe the version of the code in this field. Different code types may not be compatible with different versions. An example for this field could be Java bytecode version 1.5 which could be coded as 101.

Control bits. Encoded in SDNV format, different bits are included for different purposes. These bits have the following purposes:

- **Compression algorithm (C).** If code compression is used this bit is set.
- **Reference/Value bit (R/V).** Routing algorithms may be a source code for the code method (bit set to 0) or a reference to the routing

information tree (RIT) (bit set). The RIT path will be indicated in the RIT path field.

- **Copy Routing code bit(CP).** If set, this field indicates that the content of the code block data field should be copied to the absolute path indicated in the RIT path field.
- **Custodian bit(CT).** If set, code will only be executed if the node belongs to any of the endpoint identifiers present in the *EID-references* list.
- **Reserved.** Additional fields may be defined for future use.

RIT path. Encoded in SDNV format, this field is used to indicate a RIT path for copying or routing purposes.

Code Block data. This variable length field includes the code itself represented in the code type format, compressed or not, with the version represented in the version field. If the Reference/Value bit is set to 0, the content is the code itself. Otherwise, it defines the path inside the RIT where the code should be found.

4.3.2 Bundle Routing Code Block

The block-type code field value must be 0x10. This code will be executed by the bundle manager in order to choose among the different routing possibilities. The code takes as arguments the list of neighbours and a reference to the routing information tree (RIT). The code returns a subset of this list containing the custodian nodes where the bundle should be forwarded to, as defined in the following expression and introduced in Section 5.3.1:

$$forwardToList = f_{routing}(mess_{app,i}, neighbourList(c_j, t), rit_{c_j}), \text{ where}$$
$$forwardToList \subseteq neighbourList(c_j, t) \cup c_j$$

The bundle agent keeps the queue of the bundles waiting to be forwarded. As seen in line 2 in Algorithm 6, the *bundle agent* checks whether the first bundle in this queue includes the Routing Code Extension. If it does, this means that the bundle agent is able to execute a given method called *nextHop* which receives as arguments the list of the custodian node neighbours, the bundle destination

Algorithm 6: Bundle Agent Code. The Bundle Agent selects the most prioritised bundle and forwards it in terms of its routing code. If the bundle routing code does not select any of the available neighbours, the bundle is punished by decreasing its priority.

Input: bundleList,neighbours,RIT
Output: ∅

```
1:  function routeBundles()
2:      //check if bundle has "routing flag" set
3:      if firstBundle.hasRoutingExtension() then
4:          nextHopList = firstaBundle.nextHopList(
5:          neighbours,destination,info);
6:      else
7:          if nextHopList.intersection(neighbours)!=emptyset then
8:              forward(firstBundle,
9:              nextHopList.intersection(neighbours))
10:         else
11:             if nextHop == none then
12:                 decrease_priority(firstBundle)
13:             else
14:                 forward(firstBundle,neighbours.any())
15:             end if
16:         end if
17:     end if
18:     return ∅
19: end function
```

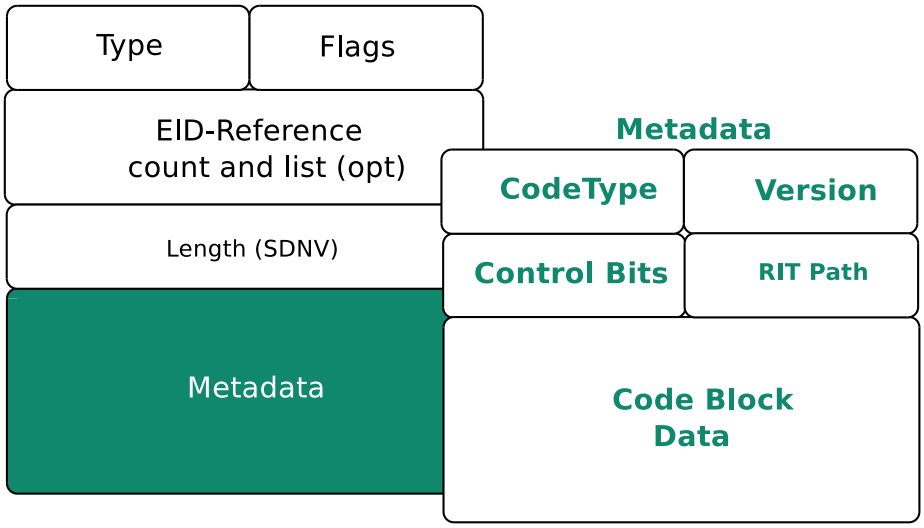


Figure 4.6: DTN Bundle with code extension block fields detailed. Metadata block is divided in several fields.

and a way of accessing the RIT and then it returns a list of custodian nodes the bundle would like be forwarded to.

A bundle routing code could eventually return *none*, and then as seen in line 10, the bundle is punished by decreasing its priority. However, not all the scenarios are appropriate for punishing bundles. This decision will depend on the different applications. By using a configurable *punish criteria* in the bundle agent, we allow different application scenarios to coexist.

4.3.3 Custodian Routing Block

The block-type code field value must be 0x11. Using this code block, application-defined codes may be installed in the custodian node. Additionally, a special field for this block must be defined:

- **Periodicity field.** Encoded in SDNV format, this field indicates the code execution frequency. The format of this field is analogous to the UNIX cron expression.

An example of this field could be *"0/2 * * * *"*. In this case the routing code will be executed every 2 minutes.

4.3.4 RIT Update Code Block

The block-type code value must be 0x12. Bundles carry code which update the routing information tree (RIT) for future bundle references. The code is executed once in every custodian node and contains write access to the RIT as outlined in Algorithm 7.

Algorithm 7: Congestion Bundle Agent Code. The *update* method retrieves a leave value from the local RIT tree. If this value is in a given range it creates a bundle with an *update extension* to inform outer DTN nodes about this fact.

Input: RIT,bundle
Output: ∅

```
1: function UPDATE
2:     value=RIT.get("./local/myapp/ttr/value")
3:     if value ≤ 20 then
4:         code=CreateCodeExtension(
5:         "./local/myapp/ttr/value",value)
6:         createInfoBundle(code)
7:     end if
8:     return ∅
9: end function
```

In Algorithm 7, the update extension code checks if locally the counter TTR (Time to Return) is about to expire. In this case it creates a bundle with an extension defined in Algorithm 8. This bundle informs the neighbours that the user is about to return to some central point so other bundles from the same application may use this information for routing purposes.

4.3.5 Lifetime Control Code Block

The block-type code value must be 0x13. The code to evaluate the bundle expiration may be included using the lifetime control code block. The function may return 0 and in this case the bundle will not be discarded. For any other value, the bundle is discarded. Further actions, such as bundle-informing the

Algorithm 8: Bundle update extension code. An example of an *update block extension* created to inform that node0 has a low *./nodes/node0/myapp/ttr* value.

```
Input: RIT,bundle
Output: ∅
1: function UPDATE
2:     //RIT update:
3:     RIT.put("./nodes/node0/myapp/ttr/value",18)
4:     return ∅
5: end function
```

source custodian node, may be taken before discarding the bundle. An example of this code could be the following:

Algorithm 9: Lifetime Control Code Example. The message reads from the RIT to find out whether a given task is finished. If it is, the message is considered to be expired.

```
Input: bundle
Output: Bundle expiration: 0/Reason
1: function LIFETIME CONTROL
2:     if get("./application/task/number/done") then
3:         bundleInform(bundle,BUNDLEEXPIRATION)
4:         return BUNDLEEXPIRATION
5:     else
6:         return 0
7:     end if
8: end function
```

4.3.6 Application Priority Code Block

The block-type code value must be 0x14. In order to allow bundles to be prioritised by using dynamic criteria, bundles may carry these criteria along with the bundles themselves using the application priority code block. An example of this code may be found in Algorithm 10.

Algorithm 10: Priority Bundle Agent Code.

```
Input: bundle
Output: Bundle priority: [MINPRIORITY,MAXPRIORITY]
 1: function PRIORITY
 2:     //Fetch state and compare it with
 3:     //application
 4:     state = 2
 5:     if state == get("./application/app/
 6:         prioritisedstate/value") then
 7:         return MINPRIORITY
 8:     else
 9:         return MAXPRIORITY
10:     end if
11: end function
```

In line 4 of Algorithm 10, the hard-coded variable *state* is statically defined in the code in the very same moment of the code creation in the source DTN node. This way, the code does not need to access payload information. The state is compared with another value obtained from the RIT to return the bundle priority. In order to modify this criterium the bundles may modify the RIT by using the RIT Update Code Block.

4.3.7 Mobile Code Implementation Details

In order to implement our proposal the JAVA-PI generic [106] implementation of the Bundle Protocol is used. It follows the Bundle Protocol Specification Version 4, available in the form of an Internet Draft [1]. The Bundle Protocol specification suggests four examples of implementation architectures, out of which our BP-RI implements a Bundle Protocol Application Server architecture.

Several changes have been performed to implement the bundle code extensions described. On the one hand, we have integrated in this bundle implementation the routing information tree. Basic primitives such as get, put, search and delete are available to bundle codes to access the Routing Information Tree (RIT).

[1] http://tools.ietf.org/id/draft-irtf-dtnrg-bundle-spec-04.txt

The RIT is structured into the different application domains that can coexist. For example, information belonging to the emergency applications are included under a different branch from those belonging to air field scenarios. Bundle's codes may access this shared structure by following the RIT, in order to retrieve the required information.

Additionally, a generic and multi-platform solution is required in order to execute mobile code. Among the different available mobile code execution environment solutions, we have used a C/C++ interpreter called Ch [26]. This interpreter supports the execution of C/C++ mobile code. This interpreter should be included in every Bundle agent running on every DTN node.

The bundle java BR-PI implementation manages the list of bundles ready to be forwarded. This list follows a FIFO model on bundle releasing. We have modified this implementation to include an application-based prioritised scheduler for ordering the custodied bundles and discarding the expired ones according to their *lifetime control code extension*. Our modified bundle agent periodically updates the bundles priorities and performs purge functions in terms of the lifetime control code described in Section 7.3.3.

Finally, we have changed the mentioned implementation in order to allow the expiration of bundles to be optionally handled by the lifetime control code described in Section 7.3.3.

In Chapter 7, an application example of this Active Message reification in the context of emergency scenarios will be explained in detail.

Part III

Scenarios

"Being practical, and considering other network evolutions such as the Internet, it would be much more useful to take advantage of such a physical infrastructure, which would allow several applications to simultaneously use the mobile nodes and their sensors in their own way."

"Lighter computers and lighter sensors would let you have more function in a given weight, which is very important if you are launching things into space, and you have to pay by the pound to put things there."

Ralph Merkle

"I can tell from the tone of your voice, Dave, that you're upset. Why don't you take a stress pill and get some rest." 2001: A Space Odyssey

Stanley Kubrick and Arthur C. Clark

5

DTN Wireless Active Sensor Networks

FOLLOWING THE architecture proposed in Chapter 3 and its mobile agent-based reification presented in Section 4.1, this chapter presents a general purpose, multi-application mobile node sensor network based on mobile code. This intelligent system can work in delay and disruption tolerant (DTN) scenarios. Mobile nodes host software mobile code with task missions and act as DTN routers following the store-carry-and-forward paradigm. Most similar proposals are unable to simultaneously run different applications, with different routing algorithms, movement models, and information retrieval strategies. The keystone of the approach in this study is using mobile code at two levels: for the application, and for the definition of the behavior in terms of routing algorithms, movement policies and sensor retrieval preferences. The intelligence of the system lies in its ability to adapt to the environment, dynamically optimizing routing algorithms using local and global information and influencing node movement. Simulations and an implementation of a real scenario have been undertaken to prove the feasibility and usability of the system, and to study its performance.

5.1 Introduction

Using a network of mobile nodes as a generalization of robots for sensing purposes is not a new idea at all. Many proposals have been described or envisaged, hitherto making use of a troop of mobile devices equipped with a bunch of sensors. Some of them even use the very same mobile node network to transmit the acquired samples to a data sink. And yet, most of these systems are oriented to just one application, and all critical mechanisms, such as message routing or the physical movement pattern, are fitted with the specific actions and main goals of that particular application.

Being practical, and considering other network evolutions such as the Internet, it would be much more useful to take advantage of such a physical infrastructure, which would allow several applications to simultaneously use the mobile nodes and their sensors in their own way. The benefits are clear: a general-purpose sensor mobile node network is easier when reusable, and it allows new applications to be implemented after the mobile node network has been deployed. Unfortunately, turning such systems into general-purpose, multiple application is not very easy. If several different applications coexist, which is the best message routing algorithm for them all? If mobile nodes can alter their pathways to adapt to some applications' requirements, what if there is a handful of applications striving against each other to make the node move under their command? These and other issues have to be considered when designing, and can be very hard to solve.

In this chapter, we propose a general-purpose sensing mobile node network, addressing the previously stated questions and many other issues surrounding the coexistence of applications and the routing of information. The rationale behind this proposal is to have mobile code at two different levels. Firstly, as the user application itself, which can move from mobile device to mobile device as it suits the application best in order to accomplish its goals. And secondly, at the message level, allowing the very same message to make the routing decision by itself, and independently from other messages' routing policies. The resulting system happens to be a particular case of Wireless Sensor Network (WSN), i.e., a network of autonomous sensors aimed at monitoring physical or environmental conditions that pass their data through the network to certain locations or data sinks [2], working as a DTN (Delay and Disruption Tolerant Networking [41]. In a DTN, data can be sent between any two nodes of the network, even when middle nodes cannot have concomitant communications. This is possible

by following an asynchronous store, carry and forward paradigm. There is a wide variety of data that can be acquired by means of sensors, ranging from temperature, humidity, or noise levels, for example, to other higher level data such as object or human recognition, movement pattern detection, or plague detection; the DTN approach only broadens the scenarios of this particular WSN, allowing a myriad of applications.

One could wonder why a DTN-based network would be better in this case than a more traditional Ad-hoc, or while we are at it, MANET, network. The main reason for this is the required density of nodes. A DTN approach, for example, does not need a figure of nodes directly proportional to the sensing area to guarantee the operation of the system. Likewise, the sensing area can be extended or shrunken depending solely on the running applications. On the contrary, communication range and sensing area determine the minimum number of nodes to be used in Ad-hoc networks. Once an Ad-hoc network has been deployed, it is very difficult to extend the sensing area without adding new nodes.

The architecture and operation of this network is presented within the chapter, which also provides details on how some issues have been resolved. For instance, how population growth is controlled when cloning is considered, how dynamic multi-routing is achieved, or how mobile messages can influence node movement respectfully and fairly to other messages. Security has been considered, as well, analyzing the threats and requirements, and giving mechanisms to protect mobile nodes.

The results are significant, and the overall performance of the system has been found very reasonable. A number of simulations have been done, from which some interesting outcomes have been drawn, such as a significant improvement in terms of message delivery ratio and latency. Furthermore, a successful proof of concept has been undertaken using real physical mobile nodes to check the feasibility of the proposal. The network has good potential for use in a variety of applications such as underwater environment sensing, unmanned aerial vehicle networks, environment applications, disaster field on emergency scenarios such as earthquakes recovery or terrorist attacks, mines seeking, urban search missions, community development, machine surveillance, accurate agriculture, biological attack reconnaissance and many others.

The original contributions of this study are: a general-purpose multi-application mobile node sensor network based on mobile code, a dynamic multi-routing schema for allowing different routing algorithms for different applications and

an application influenced movement model for minimizing node stagnation and maximizing segregation.

5.2 State of the Art

There is a wide range of literature concerning Wireless Sensor Networks (WSN) [2]. The large amount of publications revolve around the different WSN's issues, such as fault tolerance [51], scalability [16], sensor placement [52], caching [37] power consumption [74], data aggregation [58] and data gathering [77], among others.

A WSN can be also seen as a collection of different sensor nodes which coexist in scenarios in which intermittent connectivity, asymmetric bandwidths, long and variable latency and ambiguous mobility patterns can be present. There are two main projects which actively study these scenarios, so-called delay and disruption tolerant networks (DTN) [41]. The first one is the Delay Tolerant Network Research group [107], which has defined an end-to-end protocol, abstract service description for the exchange of what they call bundles [84]. These bundles carry application information from one DTN custodian node to another. The second project is Haggle [83]. Haggle defines a one-way communication architecture and its main purpose is to take advantage of brief connection opportunities. Both projects have a common aim: to propose solutions to scenarios in which network availability is intermittent or suffers from long delays by message-switching and opportunity-oriented behaviors.

Among all of the different DTN issues, routing [30] is probably the most challenging one. In this paper, we focus on dynamic and adaptive approaches for the routing problem. There are interesting proposals like [93] which include routing schemes to allow dynamic policies to choose from different routing possibilities. Authors of that proposal state, in concordance with this paper, that no unique routing solution can sufficiently cater to the different communication requirements. However, that kind of proposal is neither generic nor flexible enough to be considered for general-purpose. In the concrete case of [93], it relies solely on a three bit header which defines the different flow requirements.

DTN data mules, mobile elements which collect information from the different DTN nodes acting as routers, can be useful in the context of wireless sensor networks by collecting data from the different sensors as in [88]. These DTN data mules can be implemented with robots, mobile nodes which can be ground

or aerial vehicles. Robots can be used, as well, in the same scenarios to provide additional connectivity for disconnected networks, as in [22], where a null-space algorithm for controlling robot movement is proposed. In [95], robots are used as routers to provide communication in a wireless mesh network. They present a system design which allows robots to cooperate to improve network throughput. Another good example in the very same context is [54]: three interesting schemes are proposed to coordinate mobile robots for the concrete scenario of emergency rescue missions.

Sensor nodes can belong to different disconnected networks. While the previous described proposals accept disruptions as an idiosyncrasy of the problem, other studies like [3] and [61] propose different ways of linking the various existing partitioned sensor networks. Although these proposals may be useful in some situations, they are mainly based on adding infrastructure elements to the network. Unfortunately, this is not always feasible due to the complexity added to the system, the economical cost of the solution, or the difficulty in finding the best location for the links. Furthermore, these proposals fail when it comes to considering networks of mobile elements, such as in the examples provided in this paper.

One possible way of facing the different issues around wireless sensor networks is by using mobile code to make, autonomously and in a context aware fashion, some of the decisions regarding sensing and movement determinations. Mobile code [47] is a well-known technology supporting precisely this. Active messages are able to migrate from machine to machine and continue their execution on the destination machine. The entities on which active messages run are called execution environments. There are not many publications on studies which use mobile code in the context of DTN scenarios and wireless sensor networks. In [64], as commented in Chapter 2, a proposal is introduced. The authors sketch the idea of using mobile code which would act as *wrappers* on messages. Although they provide an interesting algorithmic approach, they give neither details about an architecture nor an implementation. Moreover, [39] presents an interesting modeling of the movement of active messages carrying application data from fragmented wireless sensor networks. Despite the extremely detailed model proposed, no implementation and no description on how to use it in a real case application is given.

There are very few studies using mobile code in DTN scenarios. In [11], the authors of this paper propose a new paradigm, *store-carry-process-and-forward*, based on mobile code to improve the integration of wireless sensor networks

and grid computing infrastructures. They describe the implementation of a delay tolerant grid service, the computer element, to give computing access to an intermittently connected wireless sensor network. The result is an intelligent system which takes the routing problem, adapts itself dynamically to intermittent disconnections and improves the coexistence of multiple grid applications.

There are interesting publications like [73], which use active messages to sense information in mobile wireless sensor networks in the context of surveillance systems. Basic primitives for active messages missions are restricted to combing instructions in specific areas, while other useful ones, such as search missions, are not considered. Routing decisions for active message forwarding are evaluated in terms of geographical information solely, while other useful information such as congestion, aggregation, movement models, or code persuasion are not taken into account. Data messages remain with the active message until it reaches a sink point, so there is no distinction between how the data messages travel and how the active messages do, which may be inefficient in some scenarios. As a result, active message missions are tightly coupled to mobile nodes, and regions with high concentrations of nodes are created. In order to avoid this, complex mechanisms must be implemented.

Even though there has been considerable ongoing work on WSN and DTN, as it has just been shown in this section, there is not yet a mobile sensor network flexible enough to fit any scenario, including those lacking continuous connectivity, and allowing to sharing the network with other applications. The next sections describe, discuss and reify precisely a system overcoming these limitations.

5.3 Proposal Description

As seen in the previous section, there is a wide range of different proposals to implement wireless sensor networks (WSN). Several issues must be taken into account to provide a general-purpose, multi-application, mobile node based WSN. These issues include sensor placement, sensor code deployment, sensing actions, sensor updating, task scheduling, routing, routing algorithms deployment, defining movement models, speed control, obstacle avoidance algorithms, energy saving strategies, data fusion/merging/overlap, data division, scheduling and dropping and finally, data processing.

In our proposal to implement a WSN, active messages play an important

role in developing these tasks. As described in Chapter 5.3.1, active messages are messages which, besides data, also carry code. We describe in this section the benefits of using active messages, which are executed on mobile nodes to obtain a general-purpose wireless sensor network. In addition to an architectural description, we explain interesting issues to be studied, as well as the limitations of our proposal.

5.3.1 Architecture

In figure 5.1, the architecture of our proposal is depicted. On one hand, we use mobile nodes which define a delay and tolerant network (DTN) by means of exploiting mobile node opportunistic or scheduled contacts. Simultaneously, on the other hand, this network is employed by active messages which are responsible for sensoring tasks. Every mobile node carries a smart device which runs an execution environment for running active messages.

Data obtained from sensing tasks may travel along with the active message as part of its data until the active message reaches a sink node. However, as shown in Figure 5.1, once data is created it can also become independent from the active message that created it. Consequently, data delivery can be considered from the traditional point of view already studied in the bibliography, as explained in Section 5.2. It has one particularity, however: the communication paradigm in this case follows a *many to one*, or *many to few* model where a small group of sink nodes represent the destination nodes. In our scenario, the deployed mobile nodes act as DTN routers, using algorithms which can be found locally in the mobile node and follow mobility models, as explained in Section 5.3.5. Section 5.3.3 describes how these routing algorithms can be updated by the very same active messages.

Figure 5.2 shows how the different actors involved in the proposal interact with each other. In section 5.4, specific technical details will be described. These actors are:

- **Mobile nodes**, robots in our proposal, carry smart devices which control sensor devices. They are capable of staying in the sensed zone for a given time and are able to come back to the **charging stations** when batteries start to run out of power.

- **Active messages** are software entities responsible for sensing tasks. They

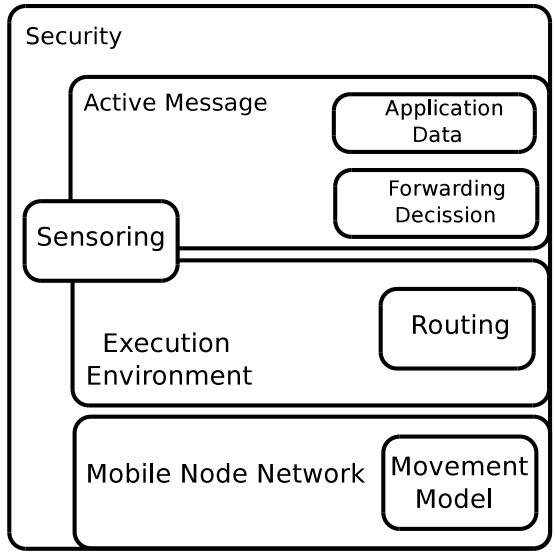


Figure 5.1: A System Architecture for DTN mobile code-based WSN.

are capable of migrating from mobile node to mobile node in order to fulfill tasks requested by users.

- **Execution environments.** Active messages need execution environments for running, which themselves run on smart devices. They are responsible for negotiating movement model changes as described in Section 5.3.5.

- **Code execution environment managers** are software entities which remain on the code execution environments. They behave as intermediary actors between hardware sensors and active messages. They are also responsible for negotiating movement model changes.

- **Data messages.** Once sensing actions are performed, data is acquired. This data has as a destination the sink nodes, responsible for the data processing. Data messages do not travel inside active messages. They are independent pieces of information that travel from mobile node to mobile node having the sink nodes as destination.

- **Sink nodes** are destination nodes for WSN. These nodes are normally stationary and are plugged, connected and located outside the sensed zone.

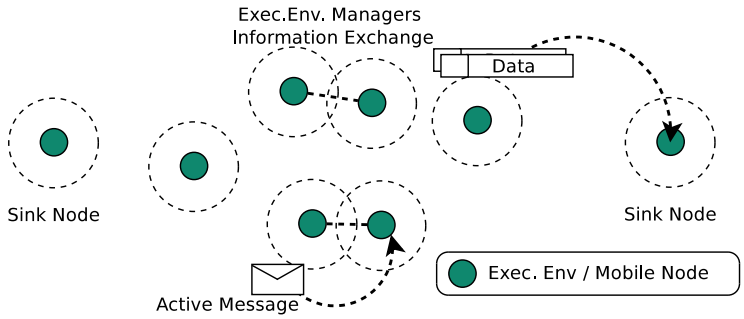


Figure 5.2: General Scenario. Active messages travel from DTN node to DTN node creating data messages which are sent to the Sink Nodes.

5.3.2 Primitive Services Types and Task Delegation

From the point of view of the user, we have identified three different potential uses for employing the network. On one hand, the user would like to search for a given pattern in a given zone in the studied area. On the other hand, the user would also like to perform a sensor comb of a chosen area. Finally, the user would like to arrive at a given point in the studied area and perform some type of unique concrete sensing. We distinguish these three types of behaviors because it will affect the way active messages will be created, how they will migrate, how they will be reproduced and their communication.

Searching active messages are source-cloned when created. This is to say that the number of copies of the message is decided before entering the affected area. Source cloning minimises communication among mobile nodes and active messages. Other proposals noted in Section 5.2 include active message migration at any given point of the studied area. However, population control is a complex issue for solving, due to its high communication requirements. Therefore, since mobile nodes suffer from limited communication and energy constraints, we see in source cloning a good solution for the population control problem.

A second service includes the types of tasks, the aim of which is to comb a given area for different sensing purposes. In this case, the active message must first approach the selected area. Once it has arrived at the selected area, the active message can then clone itself into other active messages. Control of the population is guaranteed, since clones cannot live outside the given area. This is what we call floating cloning, following a similar idea in the context of network-based social applications found in [72]. One delicate issue is that of the size of the area, that it not be large, so as to prevent the appearance of highly concentrated regions. This problem will be further discussed in Section 5.3.5 and simulated in Section 5.4.

Finally, we consider the situation in which an application wants to go to a specific place and for concrete sensing. In this case, there is no need to do any source cloning or floating cloning.

Applications, however, may want to define composite tasks containing different service types. For example, an application may want to define a search behavior in which once the active message finds its target, it can comb the surroundings for additional information. We distinguish then, among behaviors. That is, which of the three different service types the active message is following and also the tasks, the high-level instructions defined by the users.

5.3.3 Dynamic Multi-Routing

In our proposal we have to differentiate among two types of routing. On one hand, active messages jump from node to node, in order to physically arrive at different *points of interest* to perform their tasks. This action can be considered a routing decision, in the sense that the active message makes a decision among the potential neighbours, considering, as well, the possibility of staying in the current node.

We see this routing as part of the active message model, that is, a complex composite movement model made up of the list of all the n different mobile nodes the active message migrates to, as described in:

$$\text{movement}_{agent} = [m_{c_0}, m_{c_1}, ..., m_{c_n}]$$

On the other hand, sensor data retrieved by the active messages travels heading towards the sink nodes by jumping from mobile to mobile node. In scenarios in which different applications coexist, a single routing algorithm may not be enough to handle all of the applications' needs. This is why routing

algorithms must take into account information from the application layer, as well.

This leads us to the necessity of having different routing algorithms on the mobile nodes for data routing purposes. Stored locally on the mobile nodes, the routing algorithms are chosen depending on the application. Given an application message *mess*, and application *app*, a local custodian node c_j and its Routing Information Tree (RIT_{c_j}). This other routing can be expressed as:

$$forwardToList = f_{localrouting}(mess_{app_i}, neighbourList(c_j, t), RIT_{c_j}), \text{ where}$$
$$forwardToList \subseteq neighbourList(c_j, t) \cup c_j$$

That is, routing decisions are made, taking into account the list of the current neighbours and the context information.

There are many proposals which define very efficient solutions for the routing problem in DTN networks for very different DTN scenarios. However, not all applications employing the same WSN need their data to be routed in the same way. A single routing algorithm may not work for various applications. The aim of our approach is not just to improve transmission time, but also to provide a flexible and generic DTN network capable of handling different routing behaviors. The routing algorithms are easily deployed on the different mobile nodes by using the very same active message. Coexistence of different applications willing to use different routing algorithms cannot be easily deployed with other classic DTN proposals such as bundle protocol-based or Haggle-based ones. In section 5.4, some results concerning the dynamic deployment of routing algorithms are presented.

An illustrative example of the need for different routing algorithms in a single DTN scenario may be found in emergency rescue coordination applications. In disaster areas, different users such as policemen, firemen, doctors, nurses, engineers or rescue teams, among others, along with portable devices such as mobile phones or tablets, may share and use the interconnected network with a mobile robot wireless sensor network. Different applications such as victim location, notification applications, fire coordination and pollution measurement or radiation location may need different ways of information routing, scheduling, dropping or aggregating. For example, information that contains a notification to a given user or mobile node may be optimally routed using a probabilistic routing algorithm such as Prophet [30]. Alternately, routing decisions for information resulting from sensor tasks may depend on the level of importance of

the information seen from the point of view of the application. If the information is important an epidemic routing algorithm will be used. Instead, it can be discarded if the information is obsoleted by some other present in the same node.

5.3.4 Aggregation, Scheduling and Dropping

Previously, in this chapter, we have commented on the advantages of dynamic multi-routing for both levels of routing presented: active message migration and data routing. Traditional routing protocols follow address-centric protocols, in which decisions are based on the destination address. However, other proposals like [58], also consider the fact that data can be opportunistically aggregated and consolidated at the routing nodes, reducing the impact on the network. Data-centric routing protocols, that is, protocols which make decisions in terms of possible future data aggregation or data consolidation, are being considered in our proposal. In the same way, data routing could be influenced by future nodes storage congestion and more concretely by potential scheduling or dropping policies.

When facing a multi-application schema for a general-purpose wireless sensor network, data aggregation is highly correlated with the different applications. It is quite complex to provide a general solution for the aggregation problem for the different routing algorithms of the various applications. The objective is to make routing algorithms aware of potential future aggregation, dropping or scheduling actions. Therefore, a combined approach of application-based and context aware routing algorithms is crucial for these purposes. We are employing the dynamic multi routing application explained in the previous section to include in the different routing protocols ways of obtaining information about aggregation, scheduling actions and dropping policies.

5.3.5 Application Influenced Movement Model

As introduced in Section 5.2, mobile nodes on a WSN may follow different mobility models. There is a *one-to-many* relationship among a mobile node and an active message; that is, active messages are allowed to congregate in a single mobile node. In Section 5.2, we comment that other proposals like [73] suggest that both actors should be tightly coupled in a *one-to-one* way. Applications which work in concrete areas could create highly concentrated regions, which are

difficult to recover. Our proposal contends that such movement model should be as independent as possible from the rest of the mobile nodes, in order to minimise energy consumption. From a global perspective, we intend to make mobile nodes remain as widely spread out on the affected area as possible but favouring the mobile node contacts. These constraints are affected by the active messages and even by the application data which can subtly modify the mobile node movement model for task accomplishment purposes.

This movement negotiation is handled by the code execution environment managers. Active messages make movement requests to the code execution environment managers, which in turn evaluate these requests in terms of the positions postulated. A queue in which these requests are stored is defined locally in the node. To accept the movement change and therefore introduce a new point in the mobile node queue, the point requested must be inside a circle radius of kd, where d is the distance from the current position of the mobile node to the previously scheduled waypoint. The bigger the factor k is, the more tolerant to mobility the mobile node will be. The mobile node will retrieve the closest position value from this queue until this queue is empty and then proceed with its movement model.

As an example, in figure 5.3, a code execution environment manager receives a request from three active messages to temporarily modify its movement model. On the lower part of the figure, we see how the movement model is modified by the active messages.

In section 5.4, several tests are presented to understand the impact of movement model tolerance.

In order to prevent a possible mobile node stagnation caused by queued points which remain far from the mobile node, requests from the active messages can be rejected. In figure 5.4, an active message performs a movement request, for example, *Point of Interest 3* in figure 5.3. The code execution environment managers decide to accept it because it is inside the radius kd. Once the mobile node has visited the other two points of interest, the queued *Point of Interest 3* remains outside the new circle radius kd. The code execution environment manager informs the active message that its request is rejected and the latter migrates to another mobile node, as seen in Figure 5.4

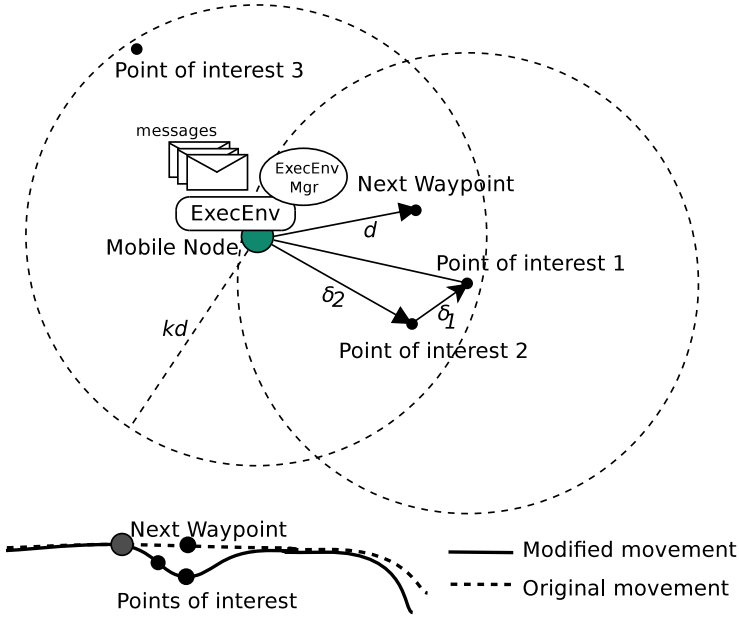


Figure 5.3: Movement model. Active messages request the local node to accept a movement change by suggesting one or more points of interest.

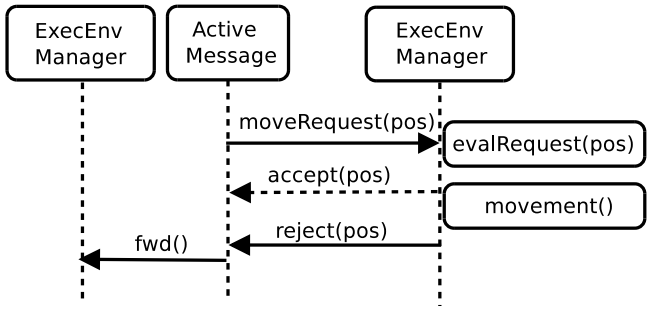


Figure 5.4: Movement request protocol. An active message requests a point of interest while the local DTN node may or not accept this movement change.

5.3.6 Active Messages: A Distributed Sensing Infrastructure

Our goal is to provide a general-purpose WSN, multi-application system. Different applications are represented by distributed streams of active messages performing different tasks in the studied zone. In this section, we enumerate these

different tasks and analyze the benefits of using the active messages paradigm in the context of WSNs.

Active messages carry the code responsible for managing the sensor retrieval following users' requests. Tasks may be divided in sub-tasks, which can be delegated to other active messages (clones) outside or inside the code execution environment, as seen in Section 5.3.2.

However, applications may at some point evolve in their needs. In a traditional WSN, this evolution means sophisticated deployment actions in order to modify the way the information is gathered, which are not always easy to achieve, as seen in Section 5.2. These adjustments may be caused by the application's need to retrieve information from additional sensors, discontinuing some others not needed anymore by the application or even modifying sensor retrieval a posteriori actions.

Active messages are an excellent option to solve these problems. Since the sensor code is travelling with the very active message itself, it is guaranteed that the way the information is gathered, plus a posteriori actions if needed, will be the ones desired by the user. Updates on these codes are made automatically if requested by the application, while visiting the sink nodes or while opportunistically contacting other active messages in the studied area.

The application itself knows best how its data should be routed. As seen in Section 5.3.3, information retrieved from the mobile sensors travels from mobile node to mobile node in data messages, following a behavior pattern set by the different routing algorithms installed locally on every mobile node. When a new application is willing to use the sensor network, it is possible to include an adhoc routing protocol for the application. Active messages carry new routing algorithms to the different mobile nodes they visit so that future bundles of data may use them. This deployment of routing algorithms may be seen as additional tasks which the active message must accomplish.

Nodes when finding other mobile nodes exchange context information which will be used by the different routing algorithms, for both active messages and data messages. Willing-to-migrate active messages communicate with a code execution environment manager to query for foreign context information in order to make routing decisions.

In Algorithm 11, the behaviour of the active message is described. Firstly, the active message completes its sensing or code updating tasks. Following, a routing decision is made on the basis of an internal method called *routing()*.

Finally, it collects the list of positions required by its tasks and proposes movement changes. In Algorithm 12, the data message behavior is described. The routing algorithm is fetched from the message itself and executed. In Algorithm 13, the Execution Environment Manager comportment is described. Context information is exchanged with the contacted neighbours and movement model requests are validated.

Algorithm 11: Active Message Algorithm.

```
1: // Task behavior
2: while task=activemessage.nextTasks() not null do
3:     if task.isMsg then
4:         //It is a sensing task
5:         msg=sense(task)
6:         send(msg)
7:     else
8:         //It is an code updating task
9:         //Update local code indicated in task
10:        updateCode(task)
11:    end if
12: end while
13: // Forwarding behavior
14: neighbours=neighbourDiscovery()
15: if neighbours > 0 then
16:    bestMobileNode=routing(neighbours)
17:    if bestMobileNode not thisMobileNode then
18:        msg.forward(bestMobileNode)
19:    end if
20: end if
21: // Movement behavior
22: if positionList=getMovement(taskCollection)>0 then
23:    sendPositions(positionList)
24: end if
25: receivereject(position)
```

Algorithm 12: Data Message Algorithm.

```
1: if msg.dest == localNode then
2:     process(msg)
3: else
4:     if neighbours > 0 then
5:         routing=getRoutingFromApp(msg)
6:         bestMobileNode=routing(neighbours,context)
7:         if bestMobileNode not thisMobileNode then
8:             msg.forward(bestMobileNode)
9:         end if
10:    end if
11: end if
```

Algorithm 13: Execution Environment Manager Algorithm.

```
1: // Context behavior
2: if neighbours > 0 then
3:     contextinforExchange()
4: end if
5: // Movement model behavior
6: if movementRequests()>0 then
7:     accepted=getAccepted(criteria)
8:     inform(accepted)
9: end if
```

5.4 Results

In order to prove that our proposal is feasible, useful and performs well in comparison to others we have several simulations have been carried out to understand issues described in Section 5.3.

We have performed several simulations [1] using *The ONE* [55] simulator to better understand several issues covered in the previous sections. In Appendix

[1] Simulations include 24 hour tests for 6 different applications running on a 5 km^2 surface with 50 nodes which create 20 KB. on average messages with different intervals from 1 second to 100 seconds. Node buffer size varies from 1 Mb. to 20 Mb and the maximum node speed is 10 m/s.

A, the classes needed to simulate active messages over mobile nodes in the context of WSNs using *The ONE* simulator are described. This includes a class caching for already visited routing algorithms for performance improvement, a movement model which simulates robots moving on the studied area, additional fields for messages description, such as the size of the active messages and payload, the *DynamicRouter* class which handles different routing algorithms behaviors dynamically and a task class for the different services types described in Section 5.3.

As commented in Section 5.3, data messages travel independently from the mobile node where they were created to a sink node. We explain in the same section, that these data messages are being routed using local algorithms deployed by the very same active message. In Figure 5.5, the result of a modified *The ONE* report[2] which produces a Google's motion chart[3] is depicted. In the figure we can see four different applications which have used our WSN. On one hand, the applications with suffix *bundle*, their data messages are routed using the traditional Bundle protocol [84] with a single routing protocol. On the other hand, the applications with suffix *dyn* each employ a different protocol chosen by the application itself. Our dynamic approach is significantly better in terms of the number delivered (circle area), latency (x-axis) and delivery ratio (y-axis) for the four applications.

We have pointed out in Section 5.3.5 how the very active message can influence this movement. The issue discussed here is which should be the movement by default, without taking into account the active message. We have implemented a model to simulate the mobile nodes' movement. The aim of this movement model is to keep all the mobile nodes in the area as segregated as possible. Mobile node movement model is almost independent from other mobile nodes. There is no geographical protocol involved which would increase the communication among the nodes, and therefore it therefore wastes energy resources needlessly. Instead, mobile nodes move following the classical Randomwaypoint, with an important modification – if n mobile nodes congregate they stop to easily exchange their data. After a given time, which will be covered in this section, nodes continue their movement in opposite directions, sharing the 2π possible directions equitably, following the behavior described in Algorithm 14

The amount of time to stop when mobile nodes contact, that is, the *rendezvous time*, is an issue we have accurately studied. It is a good idea to let the

[2] http://deic.uab.es/∼cborrego/MessageStatsMotionChart.java.v1
[3] http://tinyurl.com/googlmotion

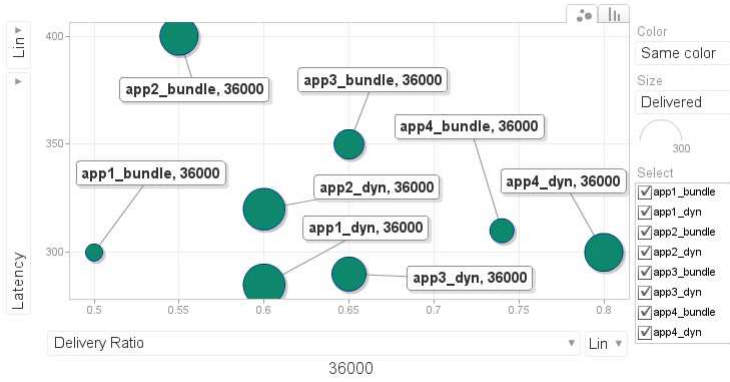


Figure 5.5: Dynamic routing versus traditional routing. The number of delivered messages is represented with the circle area. The latency time, expressed in seconds is represented in the x-axis. The delivery ratio is expressed in [0-1] values and represented in the y-axis.

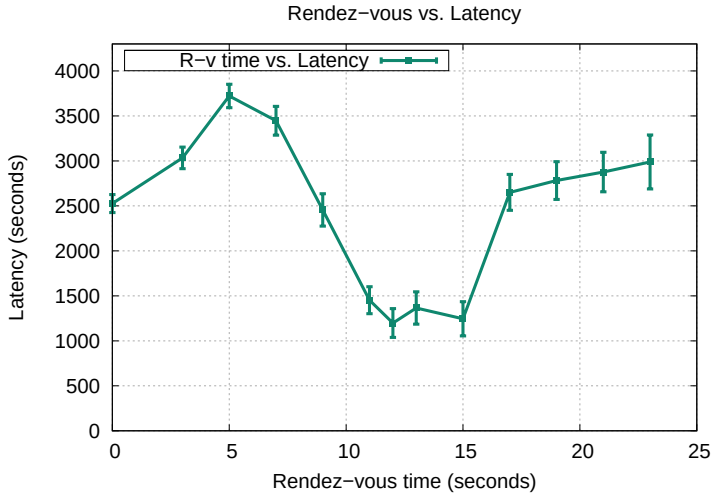


Figure 5.6: Latency time as a function of the Rendez-vous time.

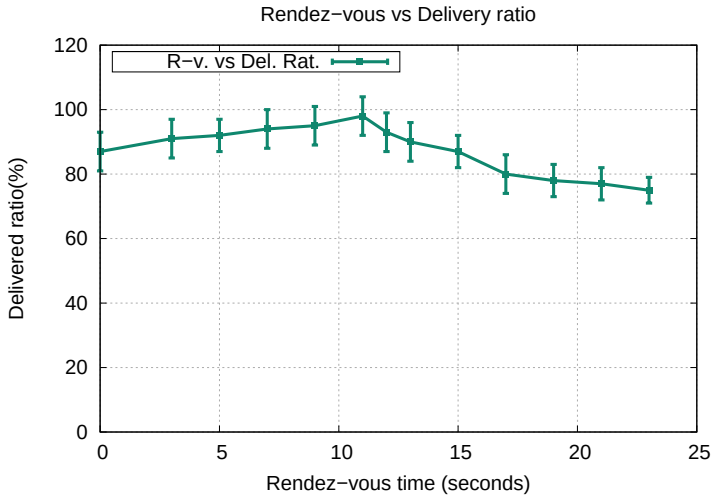


Figure 5.7: Delivery ratio as a function of the Rendez-vous time.

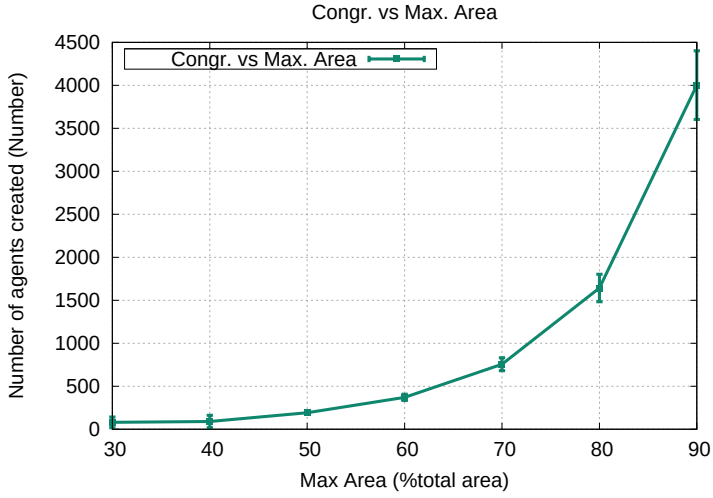


Figure 5.8: Number of agents created as a function of the size of the population control area.

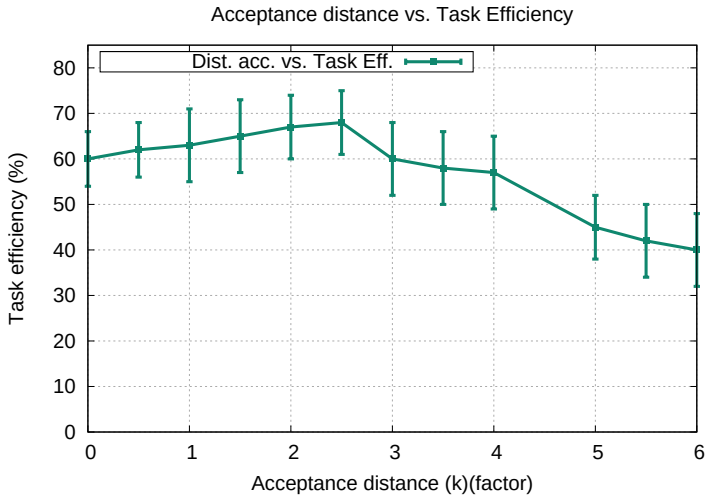


Figure 5.9: Task efficiency as a function of the acceptance distance (k factor).

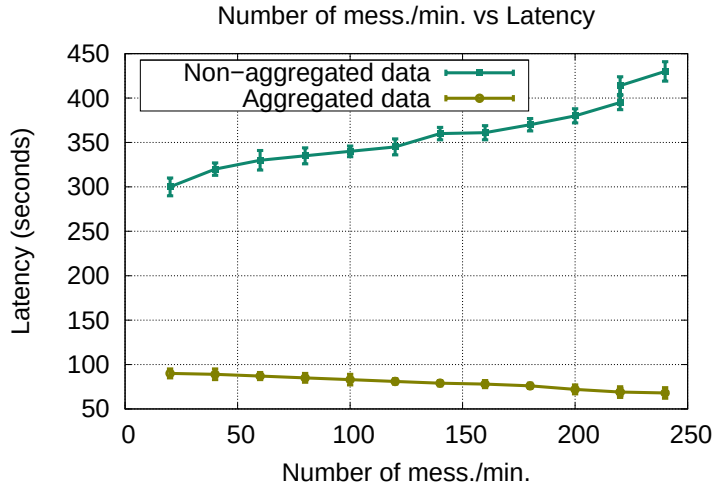


Figure 5.10: Latency time as a function of the number of messages. Aggregated data improves significantly the latency time.

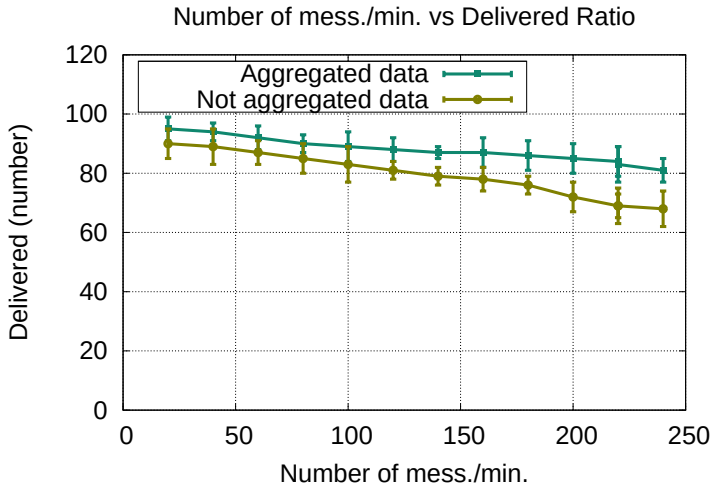


Figure 5.11: Delivery ratio as a function of the number of messages. Aggregated data improves significantly the delivery ratio.

Algorithm 14: Movement model code

```
1: ///Share 2π angle among the available neighbours
2: angle=getEquiteAngle(neighbours);
3: x = lastWaypoint(x) + maxdistance * cos(angle);
4: y = lastWaypoint(y) + maxdistance * sin(angle);
```

mobile nodes exchange the information they need, however, long *rendez-vous* times can lead to unwanted consequences. As seen in Figure 5.6 and Figure 5.7, several simulations in which different scenarios interrupted after 24 hours of *events time* are studied. *Rendez-vous time* seems to be an improvement on both delivery ratio and delivery latency. There is a point, however, at which it is not worth finishing the communication with the contacted node, as seen in both figures.

Aggregation is described in Section 5.3.4 as being very useful in Wireless Sensor Networks. Policies for aggregating information are complex as described in Section 5.3. However, if routing algorithms are deployed according to applications' constraints, information needed to aggregate, drop and/or schedule can be included inside these routing algorithms. Our proposal fits perfectly here because of its dynamism. The more different applications we get on the same scenario, the more useful our proposal is. In figures 5.10 and 5.11, we depict a scenario in which different applications coexist. Local routing protocols containing information about how to aggregate data messages are deployed by the active messages. We study delivery ratio and delivery latency in terms of the number of messages created by the active messages per minute. We compare both alternatives, using aggregation and without using aggregation. The result is a significantly improvement in terms of delivery ratio and delivery latency.

In section 5.3, we suggest a controlled cloning for task delegation, in which population control is guaranteed, since clones do not live outside the given area. The size of this area is extremely important as depicted in Figure 5.8. For a scenario with an initial number of 40 active messages, 10 robots and 40 different tasks, we can see that population grows linearly, however, at some point when the area allowed arrives the 50%, the population starts growing exponentially, making delivery ratio and delivery latency collapse.

Following the issues described in Section 5.2, in Section 5.3.5 we propose limiting the acceptance area for movement changes in terms of a circle radius of kd, where d is the distance from the mobile's node current position to the previously scheduled waypoint. In Figure 5.9, we evaluate the impact of k in terms of task efficiency (% of successfully finished tasks). In this case, efficiency improves linearly when k is increased, but because of the creation of highly concentrated regions, large areas may reduce significantly efficiency tasks results.

"All the fingerprint paintings are done without a grid."

Chuck Close

"Bicycling is a big part of the future. It has to be. There's something wrong with a society that drives a car to workout in a gym."

Bill Nye, The Science Guy

6

DTN Active Sensor Grids

STORE-CARRY-and-forward Delay and Disruption Networking protocols offer new possibilities in scenarios where there is intermittent connectivity, asymmetric bandwidths, long and variable latency and ambiguous mobility patterns. In this chapter a new paradigm – *store-carry-process-and-forward* – based on the DTN architecture presented in Chapter 3 and its mobile code-based reification described in Section 4.2, is proposed in order to improve the integration of wireless sensor networks and grid computing infrastructures. It is described the implementation of a delay tolerant grid service, the computer element, to give computing access to an intermittently connected wireless sensor network. The result is an intelligent system which adapts dynamically to intermittent disconnections and improves multi-application coexistence. Finally, it is presented as an example a real case application which provides general purpose grid access to a multi-application mobile robot node sensor network.

6.1 Introduction

Grid computing [102, 46] has consolidated as a technology capable of solving some of the most challenging scientific projects of our century. The needs of these projects usually include complex computation of data obtained from different sources and stored in large storage resources. The main goal of grid computing, precisely, is to share these resources among different institutes and virtual organizations across high-speed networks and distribute and coordinate its processing.

Wireless sensor networks (WSNs), on the other hand, is a technology that can be very useful when it comes to acquiring and transporting data collected in widely spaced areas. These networks consist of different nodes carrying different sensors along with autonomous computational devices which transmit data through the network to some specific locations or data sinks.

This Chapter analyzes how both technologies, grid computing and wireless sensor networks, can be combined into an integrated WSNs and computer grid infrastructure allowing new functionalities. The corner stone of this conjugation is using delay and disruption tolerant networking (DTN) concepts [41] along with mobile code to create an intelligent grid network capable of routing and managing processes depending on the context. Some other recent proposals, which will be further described in section 6.2, integrate WSNs and grid computing, as well. However, our proposal comes from the network perspective. We consider WSNs nodes as intermittent connected nodes, containing asymmetric bandwidths, long and variable latencies and ambiguous mobility patterns. This new perspective contributes to the creation of a novel concept of intelligent grid computing networks, going beyond the possibilities of the reviewed literature in some current scenarios, and providing promising prospects for supporting future grid services.

The routing decision making and execution policies travel with the messages, instead of being static and exactly the same for all nodes. These policies, in the shape of mobile code, can take into account the context of the nodes to choose the behavior that fits best in each situation. All in all, the system acts more like an ant colony, with differentiated autonomous parts acting locally but with a cooperative aim, rather than a traditional and more inflexible system. Thus, using mobile code makes the grid network an intelligent system, pliable enough to adapt to new scenarios of grid computing. However, the proposed system cannot be considered a silver bullet for all grid computing; in highly

connected grids, with low latencies and where data does not need to be processed before getting to the execution destination, mobile code would introduce an unnecessary extra overhead and other unwanted side effects.

6.2 State of the Art

There are several efforts already published on the integration of wireless sensor networks and grid computing. Studies like [90] and [66] propose different ways of extending the computing grid paradigm to allow the integration of wireless sensor networks and grid computing infrastructures. This section analyzes the state of the art of other technologies: mobile code and DTN protocols which we believe can extend and improve this integration. In Figure 6.1, research overlapping of the four technologies is depicted.

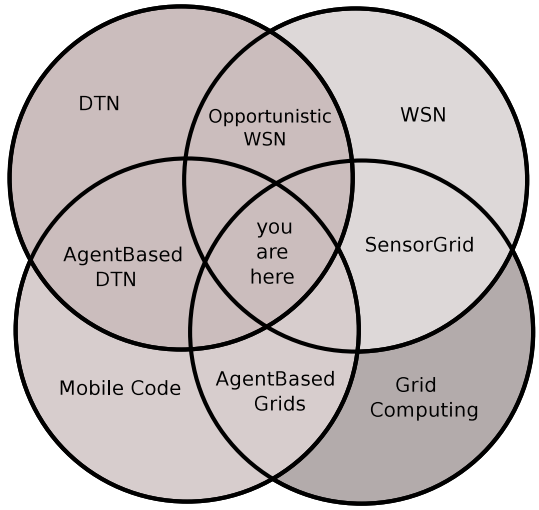


Figure 6.1: State of the art.

The widespread use of portable devices, generally equipped with wireless enabled communications, GPS receiver, and touch screen, has remarkably improved outdoor applications in a great variety of scenarios. Although the most common network configuration is adhoc, or mobile adhoc (MANET), new communication paradigms are emerging to fill the void for some specific settings.

This is the case of delay and disruption tolerant networking (DTN) [41], extremely useful when no concomitant network links connect source and ultimate destination at transmission time.

Unfortunately, although DTN has strong foundations and many groups have been working on its formalities for some years [107], [108], there is still a number of issues to be solved. Several interesting proposals have been already proposed and implemented which cover problems like routing [30], congestion [85], and security [5].

Some of these topics need solutions quite different from the ones used normally in the Internet. The rationale for this is that the diverseness of applications running on such limited connected networks calls for a number of different mechanisms to solve the specificities of each one. In opposition to what happens in the Internet, no general purpose mechanisms exist satisfying the requirements of all applications at once.

There are not many publications or studies which use mobile code in the context of DTN scenarios. As introduced in Chapter 2, in [64] an idea is sketched, but contains no architecture and no implementation is given.

On the other hand, there are few publications concerning grid computing and DTN networks. In [62] an integration of disruption tolerant networking with grid computing is presented. The article gives neither architecture proposal nor implementation and therefore is extremely superficial. The scope of this study is solely an interplanetary grid computing scenario.

Few articles propose using mobile code to solve grid computing problems. In [59] an environment based on mobile code for the distributed solution of iterative grid-based applications is presented. In [38], [29] and [91] three interesting grid monitoring proposals based on mobile code are presented.

In [12] the author of this thesis and his supervisor present an infrastructure based on mobile code to describe grid resources not only taking into account the resources themselves, but also other resources of the same type, in order to improve several grid services such as the information service and the monitoring service. The term grid resource relative information is introduced: information that is not only gathered from the very resource itself, but also taking into account other resources of the same type. This concept can be applied both to the grid information service and the monitoring service. They show that it is feasible and useful to publish relative information and monitor grid services using relative criteria. It is shown as well that relative information gives a more complete view of grid services, and it helps the user to choose the best resources

for their needs. Monitoring grid services with relative criteria is a good way to find malfunctioning services. In order to flag a service as problematic the proposal does not take into account just local sensors of the service but as well sensors values extracted from the rest of the services of the same type. Mobile agents are a very good option to use as an infrastructure to go through with these implementations. This infrastructure proposed is more flexible than traditional ones because of the sensors deployment. The update of the way local information is extracted becomes a trivial matter while using mobile agents, even if there is a large number of different services. There is an inconvenience though: every grid node must have installed a mobile agent platform where the agents will run.

There has been a considerable ongoing work on WSNs and grid computing infrastructures integration. However, there is not yet a mobile sensor network flexible enough to fit heterogeneous scenarios, including those lacking for continuous connectivity, and allowing to share the network with other grid applications. In this chapter we describe and discuss precisely a system surpassing these limitations.

6.3 Integrating DTN WSNs in Grid Computer Infrastructures Using Mobile Code

In this section we propose a way to integrate intermittently connected WSNs in computer grid infrastructures. On one hand, the data acquired by the WSNs can be processed and stored using traditional grid services inside connected regions. On the other hand, and a more challenging option, sensors can be considered as grid sources of data commanded by the very grid users themselves. We propose the job behavior that evolves beyond the traditional *read-process-and-storage* model – that is reading from a data source, processing and storing the result – to a possibility of having a *create-read-process-and-storage* model, in which jobs are able to create their input information.

For both ways of integration, there is a challenging issue which must be solved. Data must arrive from the WSN to nodes connected to the grid, known as sink nodes. However, connectivity among the nodes belonging to a WSN can rarely be available, due to large latencies, high error rates or even very small bandwidths. Examples of these scenarios include outer space sensors and computer facilities, and isolated environment sensors.

Traditional grid scenarios use different types of data sources which include large hadron colliders like the LHC [17], biomedical equipment or supplies for studies like [94] or even simulated data obtained from the very same grid resources in the context of the same domains. Most of them are constantly connected to the Internet or have a straightforward way to store its data to grid connected resources. Likewise, grid services are implemented based on networks with a constant Internet connection and taking for granted continuous network connection.

By considering WSNs a potential grid source of information, we analyze in this study how to include WSNs in grid computing, taking into account that they may be not fully permanently connected to the rest of the grid infrastructure. The result is to obtain what we call a *delay and disruption tolerant computer grid source*.

In Figure 6.2, we can see how grid infrastructures and WSNs technologies may coexist. On the left-hand side of the figure, a WSN is depicted connected to a grid infrastructure by a computing element, a door to the sensor network. This sensor network is then seen as grid source becoming grid-wide available. The WSN is composed by several sensors which, as a global network, are intermittently connected to the Internet and among them could have a poor connection. Some of these sensors may adopt ambiguous mobility patterns and therefore act as DTN data mules [41], sensors in movement which gather information from other sensors which act as opportunistic router nodes.

The traditional DTN scheme follows the well-known *store-carry-and-forward* paradigm [41]. We introduce a new paradigm: the *store-process-carry-and-forward*. We can benefit from the fact that in DTN the information can be blocked waiting in some DTN node which temporarily is disconnected. This waiting time can be extremely long and we can make good use of it. If the application data contains information to be processed we could make the local node process it while it waits for some other node to arrive.

Grid application information, that is, information created on behalf of a user and retrieved from the sensors, is carried inside messages from the sensors back to the computing element. Custodian nodes, that is, nodes which custody application information until it can be forwarded to another node, handle delays and disruptions. The application layer will create application data which will be included in one DTN message. These messages will stay in a custodian node

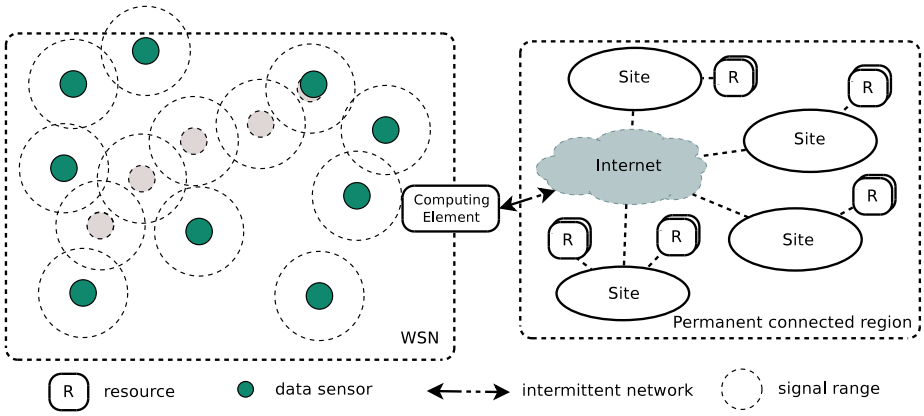


Figure 6.2: Coexistence of WSN and grid infrastructure. On the left hand side of the figure an intermittently connected sensor grid is connected to a permanent connected grid infrastructure.

until they are able to migrate to another one. Since delays in DTN networks can be huge, messages containing application data may wait for long time. Once a potential custodian node becomes available, the message will try to be forwarded to the newcomer custodian node.

As seen in Figure 6.3, traditional grid layers – application layer, collective layer, resource layer, connectivity layer and fabric layer – can be completed by adding an execution environment layer. Mobile code can be seen an alternative to the grid classic fabric layer. Its aim is to provide an interface to local resources, the sensors. There is a subtle difference with traditional grid architectures. This alternative to the fabric layer in this case is directly connected to the application layer in order to allow applications direct access to the information retrieved by the local sensors present in the nodes.

6.3.1 Grid Job Management

Information created in grid sources are processed using a computing infrastructure. Jobs input data in experiments like ATLAS [1] can reach to be tens of thousands times bigger than the output. In scenarios in which we have the grid

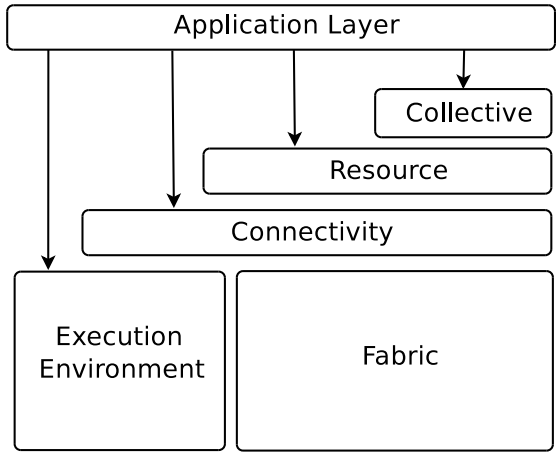


Figure 6.3: Grid layers.

sources included, such as delay and disruption WSNs, we believe that computing locally may save loads of data transfer. This is specially interesting, since these transfers are very expensive.

In the case in which WSNs are isolated, for example a sensor network in deep space, bringing data outside the network in order to compute it can be really expensive. Instead, we propose a model in which raw data, information extracted directly from the grid sources, is kept and processed locally.

Traditional grid users define their jobs by specifying their executables, their input data, their output data definition, requirements such as the site the user wants to send the job to, and the virtual organisation. The computing element receives the job specification along with the files the user has decided to send to the site. This information is passed to the different job managers. A typical job manager would receive a job request and would submit the job to a local queuing system, such as pbs [48].

In our approach for sensor grid scenarios the job manager's behaviour would be different. We have defined and implemented a computer element job manager which, instead of submitting a job to a queuing system, it will create an active message and will launch it to the WSN. Active messages are messages capable of performing processing on their own and consequently behave autonomously. These messages will travel from node to node in order to accomplish their tasks.

Creating this active message is quite a challenging issue. We need to clearly specify its behavior in terms of routing decisions, sensor retrievement, data management and task scheduling. The user, by means of the job description, can explicitly specify theses issues, otherwise, the job manager will be responsible for properly creating it.

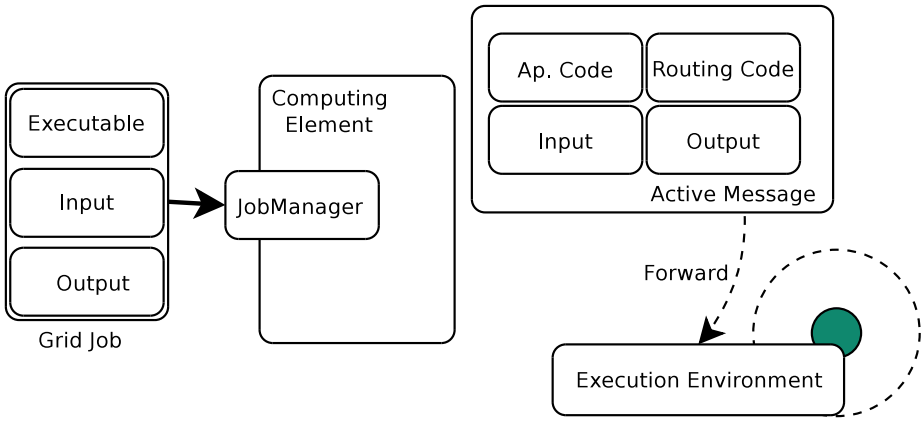


Figure 6.4: Active message creation. The grid job manager creates an active message from the grid job specification.

Looking closer at the intelligence of this message, from one hand we have to differentiate between the application code, that is the behavior the active message will require which defines the actions and tasks it will perform. These tasks include, for example, searching and combing actions in the wireless sensor network area and processing data. As a result, application data is created. From the other hand, the active message will need to make routing decisions whenever finding different nodes on the networks. Traditionally, DTN approaches leave routing decisions for DTN nodes. In our proposal, intelligence comes from allowing routing code to travel with the very active message, allowing different applications employing the same WSN to use different routing algorithms. Finally, both input and output may need to travel with the active message. This mapping is represented in Figure 6.4.

Algorithm 15: Execution environment code.

```
1: for msg in getListMessages() do
2:     //Fetch message code and input
3:     code=msg.getAppCode()
4:     input=msg.getInput()
5:     execfork(code,input,out);
6: end for
```

6.3.2 Store-Process-Carry-and-Forward Paradigm

The active message implements the *store-process-carry-and-forward* paradigm:

- Once an active messages arrives to a node, it waits **stored** in a queue of messages, until another node is available for forwarding.
- When the active message is waiting for a node to be forwarded, if the input files of the grid job are available, **process of data** can be started.
- When the message is forwarded, it behaves like a **data mule** [41] making an **active carry**, i.e. carrying information containing the grid job output. Instead, a custodian node in motion, as it could be a robot, satellite or an animal, performs a **passive carry** when moving. In this case, there is no software entity performing the transport itself. Instead, there is a physical movement of the custodian node who produces the **carry** action.
- The very active message contains internally a routing algorithm to decide when and where to be *forwarded*. The routing algorithm is part of the active message. This way we allow different applications using the same infrastructure to behave differently when it comes to routing decisions, contrary to classical networks such as TCP/IP, in which routing static elements implement and perform the routing decisions.

6.3.3 Processing Models

Bringing back the concept of data processing while waiting to migrate, it could happen that when the message is able to be forwarded from the current custodian node, the code execution has not yet finished.

We have three behavior possibilities here to define how to handle job management:

- **Run & Go.** When a new node is available for forwarding, jobs do not abort their execution. Active messages try to be forwarded once the job has finished, however, the new node still might not be available for forwarding. This option is used when node density is high. In Figures 6.5, a new node appearance is represented by an arrow. In Figure 6.5(a) concretely, a job finishes its execution and waits for next node to be forwarded.

- **Abort & Go.** Jobs could be aborted when a new custodian node appears losing all the processed information. The active message could eventually discard the computing effort already done, finding it more convenient to be forwarded to a new node. This is represented in Figure 6.5(b). Jobs will start again in the new custodian from scratch. This option is useful when contacts are scarce and processing cost is low.

- **Stop, Go & Resume.** Jobs stop their execution when they are aware of a new suitable custodian available, forwarding themselves to the new custodian and resume their processing in the new node. As seen in Figure 6.5(c), job is stopped but resumed in the new node. This option includes two possibilities which have been already described in Chapter 2.

  - *Weak mobility* is limited to the code and the data state. When arriving in the new execution environments, the code would be restarted.
  - *Strong mobility* allows code to be moved at a very low level, such as stack pointers and instruction pointers to different execution environments.

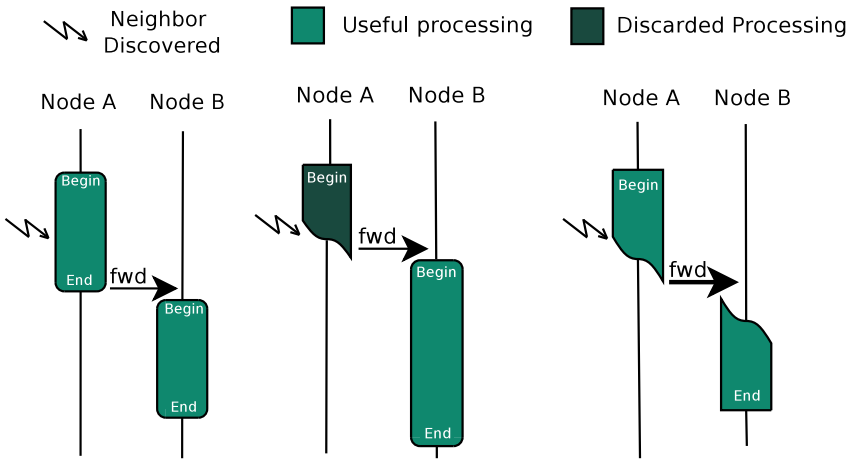
The behavior of these different models is described in Algorithm 16. Job management is done by delegating job execution to the execution environment manager (EEM). Before the active message code is invoked, its job behaviour is queried. This behaviour defines how an active message wants to proceed if the message is ready to be forwarded but the processing is not finished yet. As seen in Algorithm 15, the *EEM* is responsible for finding the application code present in the active message and for executing it. This code represents the way the active message will behave in terms of sensor retrieval and task accomplishment in a custodian node.

Algorithm 16: Routing code for the different messages waiting for being forwarded.

Input: listOfMessages,neighbours
Output: ∅

```
1:  function routeMessages()
2:      for msg in listOfMessages do
3:          //Fetch message routing
4:          routing=msg.getRouting()
5:          //Perform routing decision
6:          node=routing(neighbours)
7:          if not node == getlocalnode() then
8:              //Fetch message model
9:              mode=msg.getModel()
10:             if not mode == RUN_AND_GO then
11:                 if mode == ABORT_AND_GO then
12:                     cancel(job.getJob())
13:                     forward(msg,node)
14:                 else
15:                     if mode == STOP_AND_RESUME then
16:                         preparetoforward(job.getJob())
17:                         stop(job.getJob())
18:                         forward(msg,node)
19:                     end if
20:                 end if
21:             end if
22:         end if
23:     end for
24: end function
```

In this way, the *EEM* has control of all the computing running on the local execution environment and can stop, suspend or prioritise any computation from any active message. Jobs prioritization is not covered in this Chapter, but it is an very interesting open issue.



(a) Run&Go (b) Abort&Go (c) Stop&Go

Figure 6.5: Processing models.

6.3.4 Storage

Storage service in grid computing is responsible for managing and holding data generated by grid sources or computing jobs as well as to provide this data grid-wide upon request. In our proposal, we have two options for creating the active message concerning the storage of the information created and/or processed.

- Store the information in the disconnected region. This information will be physically stored in one of the DTN nodes from the disconnected region. The very active message will choose which would be the physical location of the data, which could be any of the DTN nodes the active message will visit, a subgroup of them or a given one.

- Store the raw data outside the disconnected region. Instead of leaving the data inside the disconnected region, the data is routed to a DTN gateway, a special node that belongs to both the disconnected region and the connected region. Subsequently, the data is stored in a storage element outside the disconnected region.

By any means, information needs to be forwarded from one node to another, traveling all around the disconnected region. Thus, the storage problem becomes a routing problem, hence, finding the appropriate DTN node to leave the data once created. Many articles such as [30] and [103] cover the still open topic of DTN routing.

In any case, the file, after being stored, needs to be catalogued so it can be located it in the future. The active message will need to reach the connected region in order to inform the file catalog about the information placement.

6.3.5 The Routing Issue

In our intelligent system, routing is the decision process in which the best path inside a given network is calculated. In traditional networks, routing decisions aim to minimise latency of message delivery and improve the delivery ratio, without overlooking local and network resource consumption. In the proposed scenario, routing is a very delicate issue. Once an active message arrives to the WSN, latency time to arrive at its destination, plus delivery ratio is important.

However, maximizing task performance in our scenario could be more important than improving latency times or delivery ratio. This performance is different from application to application and depends on several factors such as processing time, network congestion, flow control, etc. Letting the very same application perform the routing decision can improve task performance.

As an example, in Figure 6.6 an active message performs a combing task for *zone 1*. This message must choose among two potential custodians, S_2 and S_3, in order to be forwarded. Routing code for this active message chooses custodian S_2 to improve *zone 1* combing task over choosing S_3, even if this latter seems to have a direct contact with its destination. The very active message routing code prioritises task completion over delivery latency. Without an application perspective, this decision would be very difficult to make by a local routing algorithm running in a custodian node. This allows the applications to intelligently employ the network in a very *personal* way.

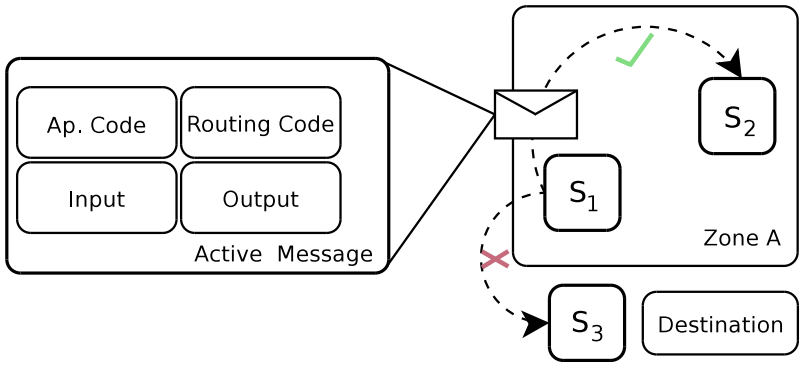


Figure 6.6: Dynamic Routing. An active message selects node S_2 as its next custodian node.

6.4 Implementation

The execution environments on which active messages run are called platforms. We need to implement on every sensor grid node an execution environment platform to let active messages carry grid level information to run on. In order to implement active messages described in the previous sections, we need some special platforms: besides being capable of executing code, platforms must allow the code to be forwarded from one sensor node to another, stopping their code, resuming its execution upon arrival and permit the ability of carrying their data and state with them.

As introduced in Section 4.2, among the different available execution environment platforms, we have decided to use Mobile-C [26], a Foundation for Intelligent Physical Agents (FIPA) [6] complaint platform for mobile C/C++ codes in networked intelligent mechatronic and embedded systems. The benefits in comparison to other execution environments like Jade based on Java are mainly achieved through the reduction in memory consumption. Moreover, in general terms, the impact on system resources is very small.

We have implemented a job manager [1] in the context of gLite grid middleware [60]. This job manager accepts traditional grid job requests and by

[1] Source code can be found at
http://ccd.uab.es/~cborrego/active-sensor-grid-src-1.2.tar.gz

parsing jobs requirements, accordingly creates active messages. We are using gLite's grid job description language to allow the user – besides the traditional *InputSandbox* and *OutputSandbox* – to specify the application code and the routing code in the creation of the active message. One example is described below.

```
JobType = "ActiveMessageJob";
ApplicationCode = "combing.cc";
RoutingCode = "myprophet.cc";
StdOutput = "output.txt";
StdError = "error.txt";
InputSandbox = {"combing.cc","myprophet.cc"};
OutputSandbox = {"output.txt","error.txt" };
```

Once active messages are created, they travel along with the grid's input data if any, from node to node, performing their tasks and retrieving information data. Execution environments custody active messages until another node becomes available for forwarding. The very same active message chooses its node path by executing the Mobile-C *MC_AddAgent()* method, as described in Algorithm 17. After which, it waits until a new node becomes available. Considering DTN scenarios, the active message can stay in this state for a longtime. We want to let active messages choose themselves their next hop, that is the routing decision. This is done by modifying the Mobile-C code to allow having a well-known method for these purposes. This method is described in Algorithm 18 and it is invoked by the local execution environment to let the active message choose among the available nodes, as seen in Algorithm 19.

6.5 A Practical Example: a General Purpose Grid Multi-Application Sensor Network

In this section we present as an example of our proposal, a way to include a general purpose multi-application mobile node sensor network, like the one described in Chapter 5, in an existing computer grid infrastructure.

We have implemented a computer element job manager which transparently gives grid access to a robot wireless sensor network. This job manager creates an active message for every grid job submitted. Users using gLite's job description language [60] choose among three different types of active message behavior

Algorithm 17: Application active message code.

```
msg = MC_ComposeAgentFromFile(
      "message", /* Name */
      "localhost:5050", /* Home */
      "IEL", /* Owner */
      "messagecode.c", /* Filename */
      NULL, /* NULL for no return */
      "nextHopApplicationLayer:5051", /* Server */
      0 ); /* no persistence. */
      /* Add the routing code to the message */
      MC_AddProcessCodeFromFile(message, "program_code.c");
/* Add the message to the execution environment to start it */
MC_AddAgent(executionenv, msg);
MC_MainLoop(executionenv);
MC_End(executionenv);
```

Algorithm 18: Active message Epidemic Routing Code example.

Input: taskList
Output: ∅

```
function PROCESS()
    for task in taskList do
        task.process();
    end for
end function
```

Algorithm 19: Local execution environment code

```
// (during the creation of the forwarding message)
if msg-→ datastate-→is_processable == 1 then
    done=Ch_CallFuncByName(*msg→msg_interp, "process", NULL);
end if
```

which will be described in this section. From the user's point of view, jobs are identical to a conventional grid job, however, input files are specified as the result of active message sensor tasks.

Concerning sensor tasks, on one hand, the user would like to search for a given pattern on a given zone on the studied area. On the other hand, a user would like to perform a sensor comb of a chosen area. Finally, the user would like to arrive at a given point on the studied area and perform some type of unique concrete sensing. We distinguish these three types of behaviors because they will affect the way active messages will be created, how they will be forwarded and their communication.

Physical tests have been performed using a conventional laptop running our modified job manager over gLite's grid middle-ware version 3.2 running Scientific Linux 5. Five real hand made chassis robots containing a PIC16C5X microcontroller, with a 4MHz processor and programmed using PBASIC. Robots carry an old Dell Axim X3 PDA which comes with 64 MB of SDRAM, 64 MB of ROM and an Intel XscaleTM processor running at up to 400 MHz. PDA batteries, lasted 253 minutes, on average.

PDAs run Windows Mobile operating system as well as the Mobile-C [26] mobile code execution environment. Some other published studies on intelligent systems in the context of traffic management and health monitoring like [27] and [28] efficiently use this technology. The benefits in comparison to other mobile code environments are mainly achieved through the reduction in memory consumption. Moreover, in general terms, the impact on system resources is very small, allowing it to be run on very limited devices.

A group of several different applications employ the network for different purposes. Robots in the tests followed a *Randomwalk* movement model. Several active messages representing the applications jumped correctly from robot to robot performing different sensor tasks. Wireless range was reduced to 15m to assure intermittent node connection. The conducted tests have value in that they helped us to obtain first impressions for network parameters such as connection window time, processing models' performance, active messages forwarding time, computer element load, job manager performance and movement model efficiency for future improved tests and simulations.

In figures 6.9, 6.10 and 6.11 three different scenarios are depicted. Space between bars represents elapsed time between messages forwarding. In Figure 6.9, a sparse scenario (less than 1 $robot/m^2$) with short jobs (less than 1 hour of total CPU usage) is depicted. Elapsed time is represented on the abscissa.

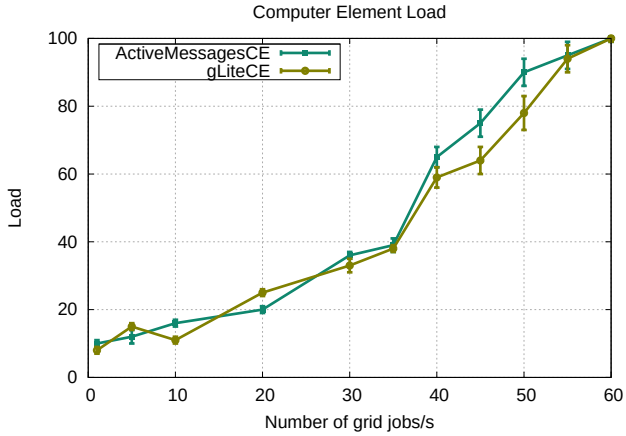


Figure 6.7: Computer element load a function of the number of jobs.

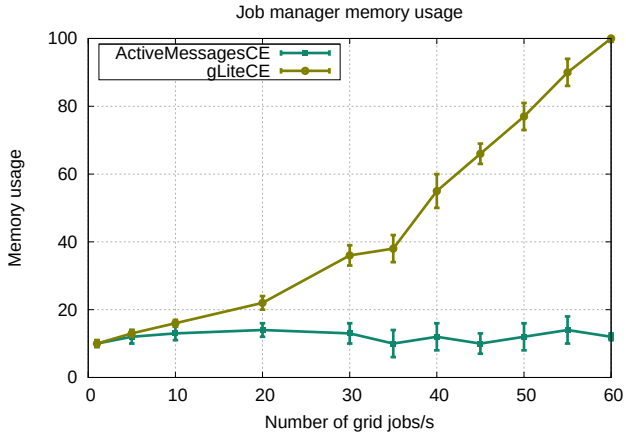


Figure 6.8: Memory usage as a function of the number of jobs.

On the ordinate accumulated data processed is represented. The three models described in section 6.3.1 equally perform for this scenario. In Figure 6.10, a scenario in which high node density (more than 1 $robot/m^2$) is present shows how *Run&Go* model performs almost as well as *Stop&Go*, while the model *Abort&Go*

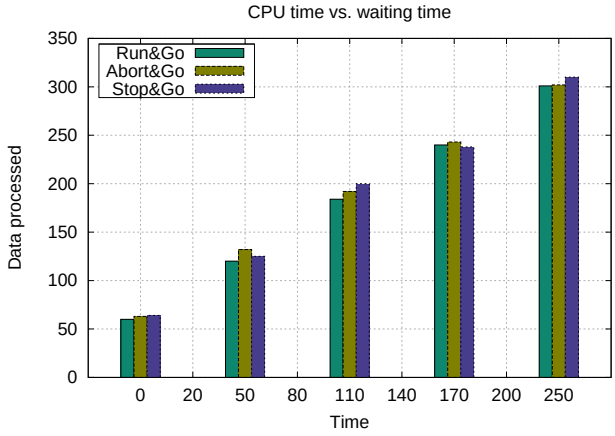


Figure 6.9: Sparse scenario with short jobs for the three different job management models.

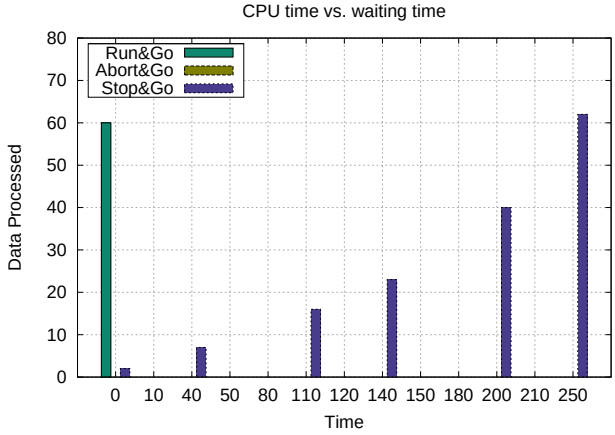


Figure 6.10: High node density scenario for the three different job management models.

is very inefficient. Finally, in Figure 6.11, we see how a dense scenario performs best when using the *Stop&Go* model.

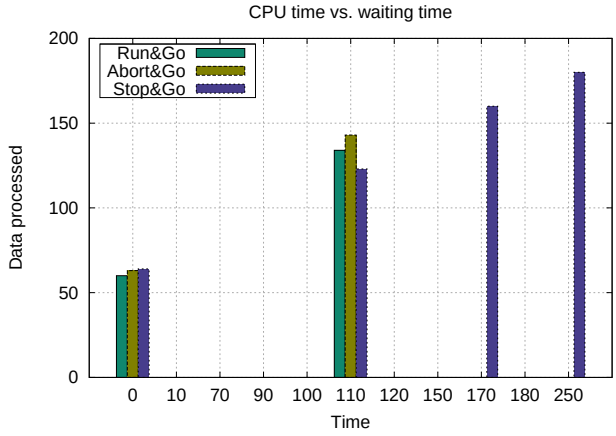


Figure 6.11: Dense scenario with long jobs for the three different job management models.

In Figure 6.7 and 6.8, the CPU usage and memory usage of our job manager in comparison with gLite's computer element is depicted. For CPU usage, we can see that similar values are obtained. Instead, for memory usage, there is a significant difference. The reason is that gLite's computer element runs a job manager process for every job for real time management purposes, while our job manager works asynchronously.

"Everybody's worried about stopping terrorism. Well, there's a really easy way: stop participating in it."

Noam Chomsky

"It's hard to believe President George Bush gave a speech in New Orleans about disaster recovery and failed to mention the word farm or the word rural."

Jim Hightower

7

DTN Emergency Scenarios

BASED ON THE architecture proposed in Chapter 3 and its Bundle Code Blocks reification presented in Section 4.3, this chapter will be introducing code to improve Delay and Disruption Tolerant Networking (DTN) performance in emergency scenarios.

Store-carry-and-forward DTN protocols offer new possibilities in scenarios where there is intermittent connectivity, asymmetric bandwidths, long and variable latency, and ambiguous mobility patterns. However, there are scenarios where current DTN arrangements are not efficient enough, such as when several applications need to coexist. In this chapter we present the application of the bundle extensions presented in Section 4.3 to allow bundles to carry mobile code and improve routing decisions. Our proposal stems from the idea of moving the routing algorithm from the host to the bundle. Bundle code may provide mechanisms to deal with congestion and lifetime control management. Additionally, bundles can be scheduled using dynamic policies exchanged and carried by other bundles for prioritization purposes to avoid important bundles from being blocked. A real case study based on an emergency scenario is presented in this chapter to provide details of a real implementation. Several simulations are

presented to prove the feasibility and usability of the system and to analyze its performance in comparison to state-of-the-art approaches.

7.1 Introduction

The widespread use of portable devices that are generally equipped with wireless-enabled communications, GPS receivers and/or touch screens has remarkably improved outdoor applications in a great variety of scenarios. Although the most common network configuration is adhoc, or mobile adhoc (MANET), new communication paradigms are emerging to fill the void for some specific settings. This is the case of Delay and Disruption Tolerant Networking (DTN) [24], which is extremely useful when no concomitant network links connect the source and destination at transmission time. Emergency and disaster recovery systems are an example of an application domain where having no network infrastructure makes DTN significantly extend possible applications. Applications based on DTN can coexist with other solutions in order to create network infrastructure and restore network connectivity. Furthermore, the DTN approach provides cheaper, easier and ready-to-use deployment solutions.

Although DTN has strong foundations such as the Bundle Protocol [84] and many groups have been working on its formalities for some years [107, 108], there are still a number of issues to be solved. Some of the most problematic issues include routing, lifetime control and security, which need solutions quite different from the ones normally used on the Internet. The rationale for this is that the diversity of applications running on such a limited connected networks calls for a number of different mechanisms to solve the specific problems presented by each application. As opposed to what happens on the Internet, no general purpose mechanisms exist which satisfy the requirements of all applications at once. A possible way of facing this challenge is by adding code to every bundle of information sent over the network to make (autonomously and in a context-aware fashion) all the decisions regarding the transportation of that particular piece of data. Mobile code [76] is a well-known technology designed for precisely this.

Mobile code is a good candidate for implementing a data-driven approach to DTN networks. This technology is able to migrate with their code and data from host to host and continue their execution upon reaching its destination. Mobile code needs execution environments to run, the execution environments.

Mobile-C [26], Jade [6] and Agentscape [98] are popular examples of platforms supporting mobile code. However, these platforms cannot be directly used for implementing DTN networks. To begin with, forwarding procedures have to be redesigned for the decision-making on which next intermediate node, also known as the custodian node, to be forwarded to. We propose leaving this decision to the bundle itself.

We present in this Chapter an extension to the Bundle Protocol based on mobile code to implement a heterogeneous data-driven DTN. While fully respecting the Delay-Tolerant Networking Architecture [18], the keystone of this integration of technologies is to move the routing algorithm from the host to the bundle. We will provide details on how this approach significantly improves these networks. We propose a scheduling infrastructure which minimises delays as well as improves reliability by prioritizing bundles carrying important application data. We describe as well several mechanisms to solve specific issues with DTN networks concerning routing, congestion, data aggregation, routing code deployment and DTN lifetime control.

Emergency and disaster recovery scenarios are very convenient to demonstrate the new possibilities of this combination of DTN concepts and mobile code. This study uses the case of the Mobile Agent Electronic Triage Tag (MAETT) [70], to illustrate how applications can profit from this new and improved architecture. However, the results can be used in many other domains.

7.2 State of the Art

There are two main developing architecture paradigms related to networks which are characterised by intermittent connectivity, asymmetric bandwidths, long and variable latency and ambiguous mobility patterns. The most relevant to this study is the Delay Tolerant Network Research group, as introduced in Chapter 2, which has defined an architecture [18] and an end-to-end protocol [84], an abstract service description for the exchange of what they call bundles in Delay and Disruption Tolerant Networking (DTN). These bundles carry application information from one DTN custodian node to another.

The second project is Haggle [83]. Haggle is a one-way communication architecture and its main purpose is to take advantage of brief connection opportunities. As in the Bundle Protocol, Haggle proposes solutions to scenarios in

which network availability is intermittent or suffers from long delays by message switching and opportunity-oriented behaviors.

While these proposals accept disruptions as the idiosyncrasy of the problem, other studies like [3] and [61] propose different ways of linking the various existing partitioned networks. Although these proposals may be useful in some situations, they are essentially based on adding infrastructure elements to the network. Unfortunately, this is not always feasible due to the complexity added to the system, the economic cost of the solution or the difficulty of finding the best location for the links. Furthermore, these proposals fail to consider networks of mobile elements such as the examples provided in this chapter.

There are very few studies using mobile code in DTN scenarios. The author of this thesis proposes a new paradigm in [11] called *store-carry-process-and-forward* which uses mobile code to improve the integration of wireless sensor networks and grid computing infrastructures. They describe the implementation of a delay tolerant grid service, the computer element, to give computing access to an intermittently connected wireless sensor network. The result is an intelligent system which takes the routing problem, adapts itself dynamically to intermittent disconnections and improves the coexistence of multiple grid applications.

Additionally, as commented in Chapter 2, in [64] a similar proposal is introduced using message relay, but the authors use an algorithmic approach and provide no details about the architecture nor the implementation. The idea of using software mobile agents is in its infancy. In [39], an interesting model is presented of the movement of mobile agents carrying application data from fragmented wireless sensor networks. Despite the extremely detailed model proposed, no implementation details are provided on its use in a real case scenario.

In addition, there are already published proposals expressing concern about effective buffer management in DTN networks. Dimitriou et al. propose in [36] a way to accelerate transmissions by saving the bundle information in memory. In [49], Henriksson et al. propose a ranking scheme using different classical caching models such as *most recently seen* and *most frequently seen* as a routing strategy. Additionally, introducing queues in the bundle layer is not a new concept. Lindgren et al. in [67] propose different strategies to drop bundles in the case the bundle cache buffer becomes full. In [57], the authors propose using information about encounters and locally collected statistics to derive an optimal policy based on global knowledge about the network.

These strategies primarily address the bundle layer information. Other layers, like the application layer containing vital information, should also be considered. Moreover, in these proposals, the criteria for bundle ordering or bundle dropping remain unchanged in each node and they do not provide any deployment mechanisms to dynamically update these criteria.

Information that is sent on DTN networks can travel from DTN node to DTN node for a very long period of time until it reaches its final destination. This time can be inadmissibly high for some applications. In the IP context, different datagrams with different types of services are treated differently. The consequences of prioritizing datagrams or not prioritizing them in the context of IP networks are not as substantial as they could be in DTN networks. While using IP's *Differentiated Services Field* we can improve datagram deliveries. Instead, in DTN networks, prioritizing DTN data units over others can be crucial to achieve the desired service.

Alternatively, Seligman et al. in [85] propose a solution to handle storage congestion. Instead of dropping bundles, they propose migrating storage data to neighbours. Other studies like [35] and [21] attempt to solve the same problem by proposing a hop-by-hop local-flow control mechanism. Both proposals only take into account local information and fail to consider data from other DTN nodes.

In relation to DTN status reports and custody signals, some efforts have been made in the context of the Bundle Protocol. Unlike ICMP, where errors and information messages are sent using IP datagrams to the source, in the Bundle Protocol as described in the Delay-Tolerant networking architecture [18], administrative records are sent to the *report-to* identifier for bundle status report, and to the identifier of the current custodian for custody signals. The criteria to send these reports and signals are static and cannot vary from application to application and do not take into account the local context.

As described in this section, there has been a considerable effort on mobile code and DTN infrastructure integration. However, there is not yet a standard solution flexible enough to fit the various necessities. The sections that follow will outline a novel approach to overcoming these limitations.

7.3 Disaster Recovery Scenarios

Disaster recoveries after emergencies, such as terrorist attacks or meteorological calamities, are difficult to conduct. Connected areas may become precipitously disconnected. Using DTN networks is an excellent way of rapidly deploying communication networks. Other studies, like [69], already use DTN networks to coordinate victims from emergency scenarios. In this section we will define a scenario based on [69] to show the advantages of using code block extensions in DTN networks.

Different users such as policemen, firemen, doctors, nurses, engineers and rescue teams, among others, along with their portable devices such as mobile phones or tablets, create the intermittently-connected network. Opportunistic contacts among the different users permit the different applications to utilise the network for very different purposes.

Rescue applications conducted by doctors, nurses and rescue teams classify the different victims during an initial evaluation of their health condition, while in the confines of the emergency area. Statuses attributed to victims are 0 for deceased, 1 for seriously injured, 2 for injured and 3 for mildly injured. This process is called *triage*. In Figure 7.1 this application is depicted.

To manage and organise injury statuses and locations we propose sending bundles with the information of every victim that has been classified. The goal of this application is to allow this information arrive as soon as possible at the *Emergency Coordination Center*, hopping from device to device when a shorter route is detected.

Other applications such as notification applications, wireless sensor applications, applications used for firefighting, pollution measurement or radiation detection could employ the same network. The coexistence of these applications is depicted in Figure 7.1. Coexistence of applications by allowing different users to share a single network decreases cost and creates a favorable atmosphere for node contacting. However, this coexistence of applications and having simultaneous users introduces some challenges that will be described in this section.

7.3.1 Dynamic Routing and Routing Algorithm Deployment

In emergency scenarios different application may coexist. For example, data from victim rescue application operations may share the network with wireless

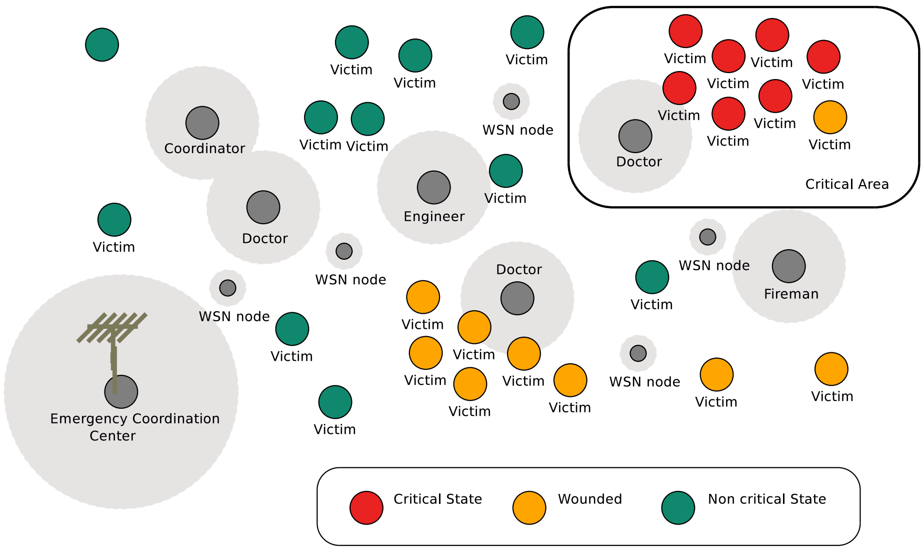


Figure 7.1: Different users in an emergency scenario create a DTN network. A sink node, the Emergency Coordination Center, collects the information from the different victims. A critical area where high density of victims is located on the top-right part of the figure.

sensor data from firefighters or other emergencies response teams. Different applications may need different types of routing algorithms. For example, information that contains a notification to a given user or mobile node may be most favorably routed using a probabilistic routing algorithm such as PRoPHET [30].

Alternately, routing decisions for information resulting from sensor tasks may depend on the level of importance of the information seen from the point of view of the application. If the information is important, an epidemic routing algorithm will be used to improve the delivery performance. Instead, it can be discarded if the information is made obsolete by other information present in the same node.

The routing decision may travel with the bundles themselves instead of being static and exactly the same for all nodes in the network, as we described in Section 5.3. These policies, in the shape of mobile code, may consider the local context to choose the behavior that fits best in each situation. Routing

algorithms are carried using the *bundle routing code blocks* described in the previous section.

We may differentiate among three types of routing algorithm deployment in emergency scenarios.

Firstly, the *Bundle Routing Code (BRC)* is the routing algorithm employed by the bundles to choose where the bundle should be forwarded to. These routing algorithms may travel along with the bundles themselves, as explained in this section. However, we propose an alternative way of deploying this code in case this paradigm cannot perform properly for a given application or a given DTN scenario.

New applications arriving to the emergency zone may perform a routing algorithm deployment before sending its data. After which, bundles may carry Routing Code Blocks with references to already deployed routing algorithms, setting the Reference/Value bit (see Section 4.3.1) and expressing the path to the routing algorithm. During the deployment phase a broadcast bundle containing a Routing Code Block with the routing algorithm is sent to the network. Additionally, the *"Local copy"* bit should be set to allow the routing algorithm to be copied locally to the Routing Information Tree (RIT) in the appropriate path indicated in the *RIT path* field.

Secondly, some routing algorithms need to periodically execute code on every custodian node. For example, a notification application for users inside the emergency area may employ a probabilistic routing algorithm such as PRoPHET [30] to route its information. This kind of routing algorithms need to execute a code on the custodian nodes to update delivery probabilities for every node contacted as well as to decrease these probabilities if the nodes are not contacted. These codes may be deployed by including a *Custodian Routing block* in a bundle. In this bundle, the periodicity of the code should be indicated by using the *Periodicity field*.

Thirdly, an application may install in the custodian nodes an aggregation function for network traffic reduction and energy consumption saving purposes using the *Custodian Routing block*. Data aggregation is an essential paradigm for DTN emergency scenarios such as DTN wireless sensor networks that sense information from the emergency area. The aim of this procedure is to merge the data coming from different custodian nodes eliminating possible redundancy, improving DTN congestion by minimizing the number of transmissions and therefore improving energy consumption. For example, different data belonging to an application from a contamination sensor network placed all along the affected area

may be aggregated following different mathematical functions such as average, summation, maximum, etc. These strategies are fully application dependant, therefore a general solution for every application employing the network will probably be inefficient.

Application routing algorithms may recognise in advance these advantages and take them into account as an extra routing criteria. Address-centric routing algorithm approaches, that is, finding the shortest path between the source and the destination can be evolved or combined with a information-centric approach in which data is consolidated by application-dependent redundant functions.

This paradigm shifts the focus from the traditional address-centric approaches of networking (finding short routes between pairs of addressable end-nodes) to a more information-centric approach (finding routes from multiple sources to a single destination that allows in-network consolidation of redundant data).

For these purposes, bundles may choose to delay its routing for a period of time in order to maximise application aggregation. Since data aggregation cannot be performed among different applications, this aggregation delay is an application related issue and may be calculated in the bundle routing code.

7.3.2 Alleviate DTN Congestion

In scenarios like disaster management, network congestion avoidance is a top priority. We consider the routing problem to be an optimal resource allocation challenge. We propose using the *RIT update code block* to allow the bundles to inform local and outer custodian nodes about different application-related issues.

Each time a user leaves the Emergency Coordination Center, they set a timer indicating when they expect to return. This timer, called Time to Return (TTR) by the authors of [70], automatically decreases its value as time goes by. If a user detects another user with a smaller TTR it will try to forward its information to this user and thus improve the expected time of arrival at the *Emergency Coordination Center*. For example, in Figure 7.2, we see how the sender application from the first custodian node chooses the lower custodian node (*doctor0*) as its *next-hop* because it has been informed by another bundle (*Bundle2*) about a low *Time To Return* to the Emergency Coordination Center. Forwarding bundles to the upper custodian node (*doctor2*) could cause congestion.

This is carried out by allowing the bundle agent to execute a concrete method

present in the code included in the bundle extension, (the RIT update code extension), as explained in Section 4.3.4.

This information will be used by other bundles to update the information on outside variables which may cause congestion, which could then be employed by their routing algorithms. Congestion is highly correlated with routing algorithms and the infrastructure proposed is an excellent way of informing the different routing algorithms from outer platforms.

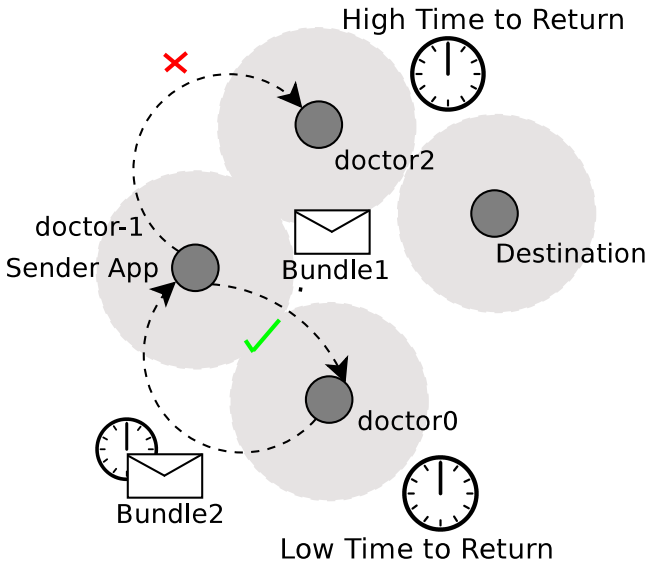


Figure 7.2: *Bundle2* forwarded from doctor-0 node carries information about doctor-0's *Time To Return (TTR)*. After reading this information, Bundle1 message on top chooses doctor-0 as its next custodian.

7.3.3 DTN Lifetime Control

In emergency scenarios, lifetime control may vary from application to application. We propose a mechanism beyond the Bundle *Lifetime* field to indicate when the bundle's payload is no longer valid. This mechanism is code-based and allows the application to control the bundle's lifetime from an application

perspective. Using the *lifetime code extension* in conjunction with the *RIT update code extension*, applications are allowed to cancel applications flows and inform both the intermediate custodian nodes and the sender application.

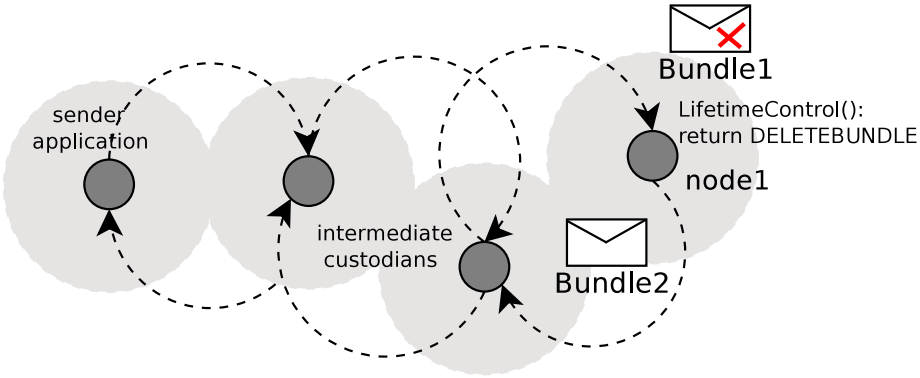


Figure 7.3: Node node1 executes *lifetime code extension* for *Bundle1*, deletes it and sends an information bundle with a *RIT update code extension* to cancel the original bundle flow in the intermediate custodian nodes and the sender custodian node.

In Figure 7.3 we see how an application sends a bundle containing some application data. Once this bundle has arrived at the fourth custodian node it executes a special code using the *lifetime code extension* to recognise that the application information carried is no longer valid. This could be due to, for example, some context change which could only be evaluated locally and from the point of view of the application. In this case, the bundle generates a control message using the *RIT update code extension* to the sender application and while visiting every DTN node it will inform all of the bundles containing application information from the same flow (if any). When the bundle arrives at the source DTN node, it will definitely cancel the application flow. Since local context information is different from one custodian node to another, we propose a solution flexible enough to allow the middle DTN custodian nodes to cancel an application flow, improving DTN congestion as bundles are cancelled.

Algorithm 20: DTN Lifetime Control Code

```
1: //Lifetime Control function:
2: function LifetimeControl
3:     //Get deadline value from the RIT.
4:     if sys.localtime > rit.get("./application/deadline/value") then
5:         createResponseBundleFromBundle(this.bundle,DELETEBUNDLE)
6:         return DELETEBUNDLE
7:     end if
8: end function
```

7.3.4 Dynamic Prioritised Scheduling

Since delays in DTN networks can be very big, bundles containing application data may pile up at the DTN layer. Once a potential custodian DTN node becomes available to a given custodian node, the latter will try to forward some of their bundles to the contacted custodian node. There is the possibility that other bundles are intending to be forwarded to the same destination custodian node at the same time. Policies like FIFO may not be enough for some DTN scenarios. To regulate these situations, a prioritised scheduler inside the custodian node is defined. This scheduler will decide bundle order forwarding and discarding policies.

In the scenarios described, while two users inside the emergency area are next to each other, bundles containing information about victims, for example, may be forwarded to another custodian node. Usually, the time it takes to forward this information is minimal due to the users being in motion. This is why it is useful to first forward bundles containing victims with a given state over others.

In the very same way, if the buffer intended for storing bundles gets full, there must be a criterium for choosing which bundles should be discarded. Proposals cited in Section 7.2 are static in the sense that all custodian nodes behave in the same way. Our proposal provides criteria for dropping bundles in the case that the buffer gets full, but with a valuable dissimilarity. We let bundles themselves carry these criteria and update custodian nodes in order to permit application dynamism when it comes to bundle prioritizing or dropping.

Our proposal consists of including bundle agents running on user's devices,

a prioritised scheduler that orders bundles while taking into account the victim's state. The prioritised scheduler must then differentiate among the varying victim states.

It is necessary to consider that the criteria used to schedule bundles could change at any given time. Let us consider the possibility of a bigger disaster which could eventually collapse disaster recovery services. In this case, disaster recovery coordinators could decide that the information on the casualties, that is victims of level 0, are no longer important. Or, in a more pessimist scenario, the coordinator could be forced to let victims of level 1 die. These criteria could change dynamically within the time of the disaster.

By using the *bundle priority extension block*, described in Section 7.3.4, bundles not only carry information about the victims, but also carry information about the criteria to schedule themselves. If the bundle agent finds that the criteria in a platform has been made obsolete by the one carried by the bundle it will update itself in the Routing Information Tree (RIT).

Bundles belonging to different applications should not compete directly among themselves. A round robin structure containing different queues for every application is proposed. The platform will go around this structure obtaining the first n elements of each queue, depending on the weight assigned to every application. This weight is determined by how important the application is and is controlled by the local bundle agent.

7.3.5 Results

In order to verify how useful our proposal is, we conducted several tests examining a concrete DTN scenario in which different users and different applications coexist in a single emergency scenario. These applications are broadcast and news messages, messages for coordinating doctors and information regarding the location and statuses of victims.

We have modified TheOne [55] simulator to perform several simulations capable of dynamically employing different routing algorithms depending on the context and the type of application[1,2].

[1]Source code for routing algorithm can be found at http://ccd.uab.es/~cborrego/bundleCodeExtension/DynamicRouting.java

[2]Simulations in this section include 120 hour tests, on a 15 km^2 area with 50 nodes. Message interval creation from 1 minute to 250 minutes. Message size from 1Kb. to 50Kb.. Node buffer size varies from 1 Mb. to 20 Mb and the maximum node speed is 8 m/s.

Users present in the affected area produce a common mobility pattern. When users leave the Emergency Coordination Center (ECC), they set a timer indicating when they expect to return (TTR). This timer automatically decreases its value as time goes by. In this case, bundles carried by a DTN node which detects another node with a smaller TTR, will forward them to another node with a smaller TTR. This will improve the expected time of arrival to the Emergency Coordination Center. Applications which do not belong to the medical domain would not understand this field and therefore would not be taken into account for routing decisions.

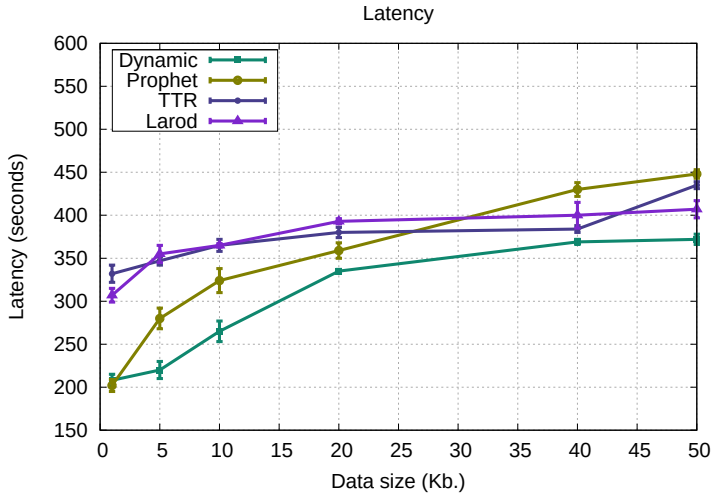


Figure 7.4: Latency time as a function of the bundle size for three different DTN routing algorithms and our dynamic proposal, that performs better independently of the size of the bundle.

Information services offer the possibility of providing notifications, news and alerts to nodes inside the affected area. These messages can be intended to affect the nodes' movement models, if for example, some emergency stuff needed to return to the ECC because of some unexpected reason. Since the users movement models present in the studied area are likely to be predictable, and depend on node locations, we expect that the notifications and alerts can be routed optimally using a probabilistic algorithm or a location-aware routing

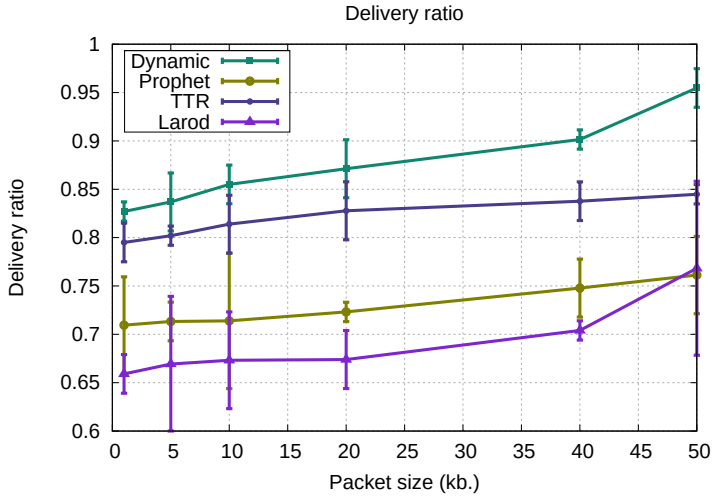


Figure 7.5: Bundle delivery ratio as a function of the bundle size for three different DTN routing algorithms and our dynamic proposal, that performs better independently of the size of the bundle.

protocol such as Larod [30], a DTN routing algorithm which combines beacon less geographical routing with the store-carry-forward paradigm.

The first series of simulations dynamically employed different routing algorithms for the four different applications enumerated before. Depending on the application, a different routing algorithm is employed. For the medical service concerning the data on those injured, a TTR-based routing algorithm[3] was used, while the notification application data used PRoPHET [30], a probabilistic routing algorithm. Broadcast and news messages were routed using an epidemic routing protocol [30]. News services may be routed as well using Larod[4] routing protocol if the news were addressed to a specific physical area, for example, an area where a fire has occurred.

In Figure 7.4 the latency average, (the average time for data to arrive from the source to the destination), is compared as a function of the size of the messages. We can see that our proposal that uses the *Bundle routing code block*

[3]Source code can be found at http://ccd.uab.es/∼cborrego/dynamicrouting/TTRRouting.java
[4]Source code can be found at http://ccd.uab.es/∼cborrego/dynamicrouting/Larod.java

performed better than the ones based on a single routing scheme. In Figure 7.5, the delivered ratio, which is the percentage of messages arrived compared to the ones sent was studied as a function of the size of the messages. Delivered ratio obtained, performed best, independently from the size of the message, if the dynamic routing protocol was used, over the rest of the simulations based on TTR routing, probabilistic routing or location-aware routing protocols.

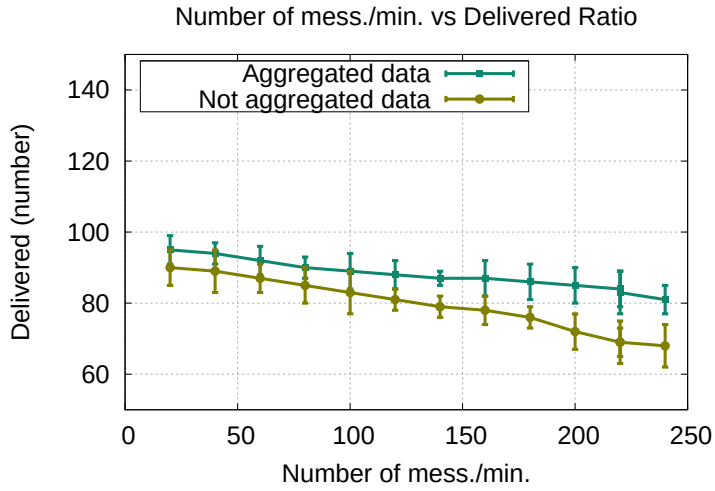


Figure 7.6: Delivery ratio studied as a function of the number of messages in a scenario that uses the *custodian block code extension* to deploy aggregation code versus another scenario which does not.

Additional simulations represent a hypothetical disaster area in which bundles chose whether to be forwarded or not to the different available custodian nodes in terms of the *Time To Return(TTR)*. This information was propagated using *the update code bundle extension* described in Section 4.3.4. In Figure 7.8, we analysed bundle arrival with two different alternatives. In a scenario in which low TTR values were propagated using the *update code bundle extension* a high bundle arrival efficiency was obtained while in another scenario where this extension was not used, the arrival efficiency considerably worse.

Additionally, we compared three different scenarios in which application priorities change at intervals of every one fifth of the simulation time. The first

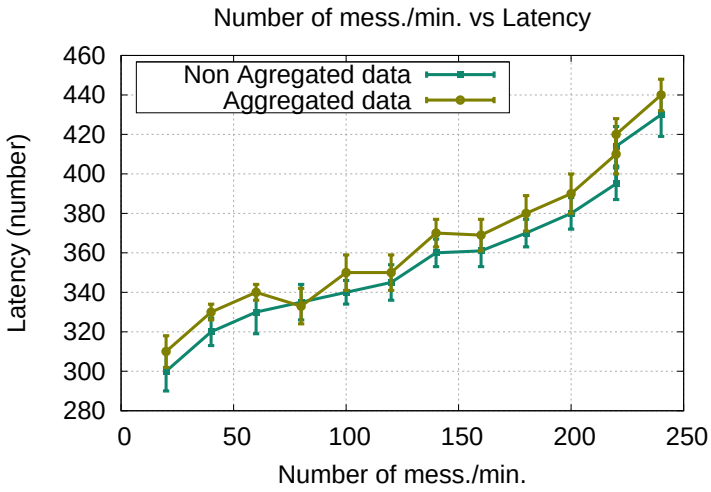


Figure 7.7: Latency studied as a function of the number of messages in a scenario that uses the *custodian block code extension* to deploy aggregation code versus another scenario which does not.

scenario did not consider bundle prioritization, the second scenario included a traditional source bundle prioritization scheme using the bundle priority field and the third one used the *priority extension block*, as described in Section 7.3.4. In Figure 7.9 we can see that our proposal performed better than the schemes with no prioritization and source-prioritization.

In figures 7.7 and 7.6, we depict a scenario in which local routing protocols containing information about how to aggregate data messages are deployed by the active messages. We analysed the delivery ratio and delivery latency in terms of the number of messages created by the active messages per minute. We compared both alternatives, using aggregation and without using aggregation. As a result of this approach, the delivery ratio and the delivery latency was seen to significantly improve.

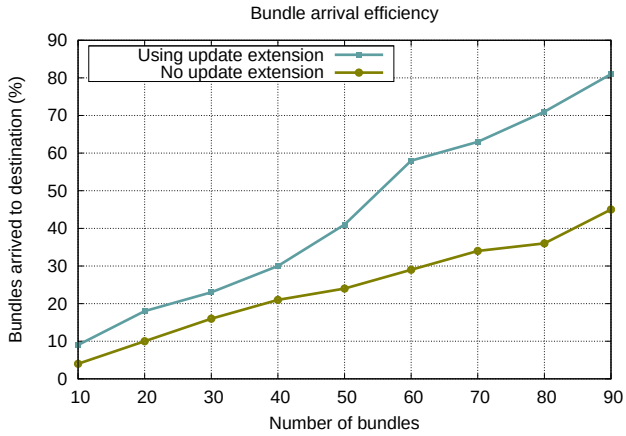


Figure 7.8: Agent arrival efficiency as a function of the number of messages. The scenario which uses the *update code bundle extension* performs better than another scenario which does not.

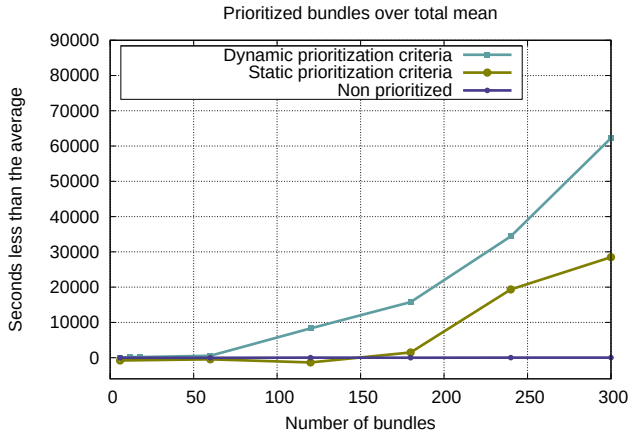


Figure 7.9: Prioritised bundles over total mean as a function of the number of messages. Dynamic prioritization performs better than the schemes with no prioritization and source-prioritization.

Part IV

Conclusions and Future Lines

"Far from being a silver bullet for all sensing applications, there is still a lot of work to do to improve this general purpose network."

"Stupidity lies in wanting to draw conclusions."
Gustave Flaubert

"Observation: I can't see a thing
Conclusion: Dinosaurs.", Cosmos
Carl Sagan

8

Conclusions and Future Lines

In CHAPTER 1, the objectives of this thesis were presented: the main goal of this study was to provide different mechanisms to improve the coexistence of several applications in Delay and Disruption Tolerant Network (DTN) scenarios by means of carrying different codes along with the data messages themselves. In this chapter, we discuss how we have fulfilled these objectives and we present future lines of research in which we are already working on. In Section 8.1, we explain the evolution of the architecture proposal in terms of the different data structure employed as well as a resume of the different applications of the proposed architecture presented in this thesis. Following, in Section 8.2, there is a brief discussion on limitations of our proposal and additional topics which still remain to be fully analyzed. Finally, in Section 8.3, future lines of study and research will be presented.

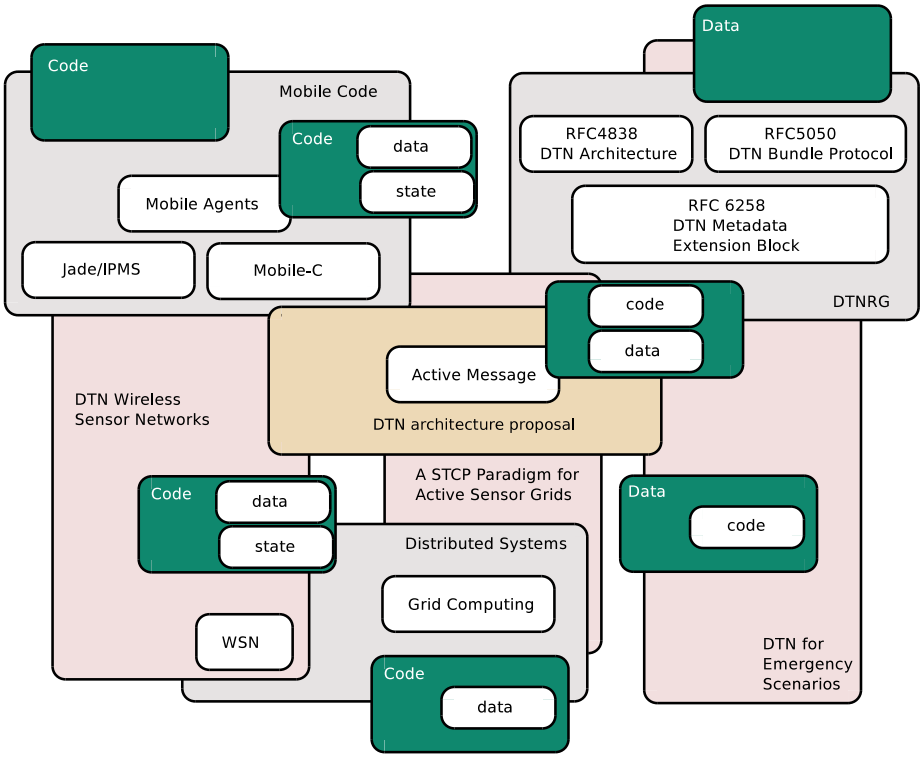


Figure 8.1: The big picture: the architecture proposal, the different reifications and their applications.

8.1 Conclusions

In Chapter 1, the objectives of this thesis were presented. The main goal of this study was to provide different mechanisms to improve the coexistence of several applications in Delay and Disruption Tolerant Network scenarios by means of carrying different codes along with the data messages themselves. In this section, we discuss how we have fulfilled this objective.

In Figure 8.1, a general diagram of the content of this thesis is presented. Depicted in grey boxes, the different state-of-the-art technologies involved in this study are diagrammed. On one hand, on the top right part of the figure,

the Delay and Disruption Tolerant architecture [18], its Bundle Protocol [84] and the Bundle Extension [89] are depicted. In this document, information about these protocols can be found in Section 2.3. Additionally, on the top left part of the figure, mobile agents, a specific form of mobile code are diagrammed. Two different types of mobile agents employed in this study are illustrated. In Section 2.5, these technologies are explained. On the other hand, on the bottom part of the figure, the Grid Computing Technology is pictured. Finally, on the bottom left part of the figure, the wireless sensor networks technology is depicted. In sections 6.2 and 5.2, the states of the art of Grid Computing and WSN concerning DTN networks are discussed.

Depicted in brown, the active message paradigm explained in Chapter 3 is diagrammed. Three different applications of this paradigm have been proposed following the three different reifications of the proposed architecture described in Section 5.3.1. The order in which the different reifications of our active message concept described in Section 3.3, is the order in which these ones were proposed. We do not discard proposing some others, as introduced in Chapter 1, however a unique solution to cover all the possible scenarios and applications is not a trivial issue.

In Chapter 5, a general-purpose sensing mobile node network that can run several applications simultaneously in scenarios with limited connectivity has been introduced. The proposal addresses a number of issues surrounding the coexistence of applications, the routing of information, and the shared use of the mobile node. The cornerstone of this proposal is mobile code, which works at two different levels: on the one hand, as the user application, the program with the sensing tasks; and on the other hand, at the message level, allowing messages to make the routing decision by themselves independently from other messages' routing policies.

The initial challenge has not been easy to overcome. There has been many issues to solve with no evident answers. However, in the end, the final network achieves its goals fulfilling all the requirements thanks to a combination of inter-related mechanisms. It is noteworthy that the dynamic multi-routing concept, is based on having the routing algorithm in the active messages, instead of in the mobile node. In this way, every application can use its own algorithm and local context information, thus optimizing global routing. Also remarkable is the mechanism allowing active messages to influence the mobile node movement, fairly and respectfully to other messages.

Different experiments were carried out with different simulations to obtain

basic parameters such as active message's migration time and signal range. It was also useful for checking the feasibility of the proposal. The simulations have shown how the different strategies used significantly improve the delivery ratio and delivery latency of messages.

The proposed network shows promises in a variety of scenarios, especially where communication is not always possible, such as environmental sensing, unmanned aerial vehicle networks, or machine surveillance and person seeking in the aftermath of wars.

This first application, *DTN Wireless Sensor Networks* in the figure, uses mobile agents to define the general purpose, multi-application mobile node sensor network for intermittently connected networks. The data structure employed to carry the application information is mobile code which is stored inside the code variables. In general terms, it is mainly code travelling from node to node in order to be executed, as depicted in the top-left green box.

On the other hand, we have also presented in this thesis an intelligent system to transparently integrate intermittently connected wireless sensor networks to computing grid infrastructures. We have presented a new paradigm called *store-carry-process-and-forward*, which proposes a way of processing data while the information is stored, waiting for the DTN information to be forwarded.

As a practical example of our proposal, we have introduced a computer element service which provides access to a general purpose sensing mobile node network that can run several applications simultaneously in scenarios with limited connectivity. The proposal addresses a number of issues around the coexistence of grid applications, the routing of the information, and the shared use of the mobile node. The cornerstone of this proposal are the active messages, which work at two different levels: on the one hand, as the user application, the program with the sensing tasks; and on the other hand, allowing messages to make the routing decision by themselves and independently from other messages' routing policies.

This second application of the active depicted as *A STCPF* (Store-carry-process-and-forward paradigm for active Sensor Grids uses as well mobile agents but in a more general way. The mobile code is not just an autonomous code which migrates from node to node, as in the previous reification. This code, instead, can be seen as a container for different codes such as routing codes and processing codes, which are launched by the local execution environment. As explained in Section 6.3.1, the code carried by the active messages may be stopped and resumed in a new custodian node. Therefore, the data structure

carried by the custodian nodes is still a mobile code which carries the application data but with a subtle difference: different code methods are launched by the custodian node for every different functionality such as routing, application process, prioritisation, etc.

Finally, we have presented a system based on the Bundle Protocol which introduces the possibility to the bundles of executing code in the very custodian nodes. We have defined several extensions for the Bundle Protocol to allow the bundles to carry code to improve different DTN issues. Five types of code blocks that may be optionally included in a bundle have been defined. Bundle code may provide mechanisms to deal with congestion, and lifetime control management, code deployment and data aggregation. Additionally, bundles can be scheduled using dynamic policies exchanged and carried by other bundles for prioritization purposes to avoid important bundles from being blocked. We have described the feasibility, utility and usefulness of this proposal by applying it to a scenario of disaster recovery systems as described in Section 7.3. We have also included in Section 7.3.5 some feasibility performance studies for different DTN applications which conclude that our proposal is valuable for some concrete scenarios.

This third application that follows the active message paradigm is depicted in the top right hand side of Figure 8.1. It uses both the Bundle Protocol described in Chapter 2 and the active message paradigm described in Chapter 3. In this case, the data structure that carries the application information is the bundle, as defined in the Bundle Protocol [84]. These bundles may optionally carry, using Bundle blocks described in [89], different function methods which work the very same way as the methods in the previous application, but without the need of using mobile agent infrastructure.

8.2 Heading Beyond

It is a fact that the complexity of the system as a whole has increased by moving the routing code implementation from the host to the bundle itself. However, the advantages of employing such a paradigm in some scenarios, such as disaster recovery systems, grid computing or wireless sensor networks, absolutely outweigh the impediments related to a system with added complexity. The aim of our approach is not only to improve transmission time, but to provide a flexible and generic DTN infrastructure capable of handling different routing behaviors and application necessities. The routing algorithms are easily deployed using the

very same bundles which carry the application data itself. Coexistence of different applications needing to use different routing algorithms can not be easily deployed with other classic DTN proposals. Our proposal overcomes obstacles which would not be easily performed with traditional approaches.

However, using a flexible architecture has its drawbacks. Every DTN node must include an executing environment, which in some scenarios might be too expensive because of computation or energy consumption reasons. In addition, there is an overhead of information transmitted since the active messages are carrying their routing algorithm in addition to the application data itself. Nevertheless, these static routing codes can be cached on the different execution environments, to efficiently manage the distribution of agent codes. Caching these active message codes improves and speeds up migrations.

We have not covered the problem of greediness when it comes to accept or forward the active messages. In scenarios like the one proposed in Chapter 5, the robot network is shared by the different applications. Fairness while balancing the use of the network resources from the different applications is easy to control since there is just one type of custodian node. However, in scenarios like the one described in Chapter 7, the different custodian nodes belong to different applications which may not always have a common goal. Credit-based mechanisms like the ones proposed in [104] should be analyzed to prevent greedy applications from monopolizing the network resources.

As commented in Chapter 3, beacon messages are sent by DTN nodes at certain intervals of time to allow neighbours to be discovered. In order to be flexible enough, we allow the very same application messages flag in order to be announced the desired RIT fields from the RIT application branch. In a scenario with many applications, this may not scale properly. It is the node's responsibility to choose among the different RIT fields to be announced taking into account variables such as average contact time, number of applications, connection speed, etc.

Although security is an important issue, security measures are not directly covered in this study. We have enumerated in Section 3.6 the security considerations which should be taken into account in our proposal. These considerations include: code confidentiality, authenticity and integrity, code confidentiality and integrity, key management, routing data authenticity and integrity and message forwarding non-repudiability. Our experience shows that these services can be obtained by different means such as by securing the bundles themselves as in [45], by using a PKI infrastructure, by using security tokens based on Identity Based

Cryptography as in [5] or by employing trust models as in [78]. Additionally, the authors of this paper have previously published studies about access control to routing information trees in the context of DTN networks. An example of these studies can be found in [81].

8.3 Future Lines

Far from being a silver bullet for all sensing applications, there is still a lot of work to do to improve this general purpose network. One of the areas in which we are working is security. In the security section, basic requirements have been described, as well as some well-known mechanisms like IBC. With the exception of the non-repudiation requirement, the other ones have been addressed in the implementation. However, this has not been taken into consideration in the simulations. We especially plan to perform some research regarding the non-repudiability requisite, since the current protocols seem to be unsuitable for our scenario. Another projected line of research is the study of the effect of having different routing algorithms in the same network on the global routing performance, and how mobile nodes could participate in the routing process as facilitators.

Further, we have been able to gain significant experience in scheduling systems in order to prioritise important information. We believe that this scheduling can be joined in the grid proposal, in order to improve services such as computing and storage. In future work we wish to propose mechanisms to exchange bundle priorities among different agent platforms to accept or deny agent immigration and to include a way to interrupt non-prioritised bundles in order to allow prioritised ones to run.

Another very interesting topic which we are intended to study is how routing application information from a given application may be employed by other applications. Research lines such as trust models or fuzzy role-based access control schemes may be very useful to apply. This issue has some security consideration which should be taken into account.

When it comes to giving a general-purpose DTN solution for different applications with different routing needs, coexisting together, routing algorithms must be dynamically deployed all over the DTN nodes. Internal optimal DTN routing algorithms parameters or application creation network variables such as the number of copies of a message to be sent, may change in terms of the context, from

Paradigm	Message	Supervisor	Adaptive	Simulation
Classic	Application data	Not needed	No	Not needed
Source Optimization	Application data Optimal Parameter	Not needed	No	Not needed
Node Optimization	Application data Optimal Parameter Code Application code	Not needed	Yes	Not needed
Sink Node	Application data Optimal Parameter Code	Needed	Yes	Not needed
Sink Simulations	Application data Optimal Parameter Code	Needed	Yes	Needed

Table 8.1: Optimization techniques.

application to application, and from DTN node to DTN node. As a future line of research we are willing to introduce several context-aware, application-based dynamic configuration procedures to improve DTN routing protocols and application creation decisions in order to dynamically obtain these internal optimal DTN routing algorithms parameters or application creation network variables.

We intend to define a set of different procedures to optimise these routing parameters. In Table 8.1, a summary of the different techniques willing to be implemented is presented. We distinguish among source-based optimising procedures from the ones on which the optimization is done on every DTN node. Our intention is to implement different approaches that may carry just the application data, optimal routing parameters or the optimization code itself. A supervisor node could be useful to have global view of the network and in an adaptive way calculate optimal network routing parameters. This supervisor node could obtain routing and application creation optimal parameters from the scenario variability, historical information, on the fly simulated data or context information.

Part V

Appendices

"Taking into account that our proposal is quiet specific, in order to simulate scenarios implemented with our proposal several changes must be done to the original The ONE simulator."

"Animals can adapt to problems and make inventions, but often no faster than natural selection can do its work. The world acts as its own simulator in the case of natural selection."

Vernor Vinge

"It (the computer) is a medium that can dynamically simulate the details of any other medium, including media that cannot exist physically. It is not a tool, although it can act like many tools."

Alan Kay

A

Simulations Benchmarks

SIMULATIONS ARE AN excellent way of testing innovate proposals. There are different simulators which represent DTN networks. Among the available ones, *the ONE* [55], provides a way of generating node movement using different movement models, defining routing messages between nodes with various DTN routing algorithms and sender and receiver types and visualizing both mobility and message passing in real time in its graphical user interface. TheONE can import mobility data from real-world traces or other mobility generators. It can also produce a variety of reports from node movement to message passing and general statistics. The most interesting issue about this simulator is that changes to its code are very easy to conduct. Taking into account that our proposal is quite specific, in order to simulate scenarios implemented with our proposal several changes must be done to the original The ONE simulator. In this section these changes are presented.

In Figure A.1, the classes needed to implement mobile agents in *the ONE* simulator are depicted. This includes a class caching for already visited routing algorithms for performance improving, a movement model which simulates doctors entering and exiting the disaster area, some fields added the messages

description, such as the size of the agent and payload and the *DynamicRouter* class which handles different routing algorithms behaviors dynamically.

A.1 Active Message Class

As described in Chapter 3, application data is carried by active messages, messages which, besides data, also carry code. These structures are not directly provided by The ONE simulator. A class which handles precisely this has been added to The ONE simulator. Application messages are carried by the active messages which also contain the routing algorithms which will be employed on every hop the bundle of information will perform. As seen in Figure A.2, three different classes have been defined to distinguish among the three different reifications described in Chapter 4.

For performance purposes, and as described in [33], these static routing codes have been cached on the different DTN node execution environments platforms, to efficiently deal with the distribution of agent codes. Caching these mobile agent codes improves and speeds up active message forwarding.

A.2 Reports

The ONE reports are very useful to understand the results of the simulations. Unfortunately, the reports provided by The ONE official distribution are not very useful when it comes to analysing the impact of dynamic routing on different applications. Since the ONE reports just give information about global network performance, a first report which provides statistics such as message creation, delivery ratio and latency time statistics for the different applications in the scenario has been defined. Additionally, in Figure 5.5, the result of a modified *the ONE* report which produces a Google's motion chart[1] is depicted. In the figure we can see four different applications which represent a DTN scenario. The report creates a four dimension graphic in which delivery ratio, latency time, number of delivered are analyzed as a function of the time for the different applications. These reports have been extremely useful to understand how the network evolves when introducing new elements such as DTN custodian nodes or new applications with different routing needs. For example, in Figure

[1] http://tinyurl.com/googlmotion

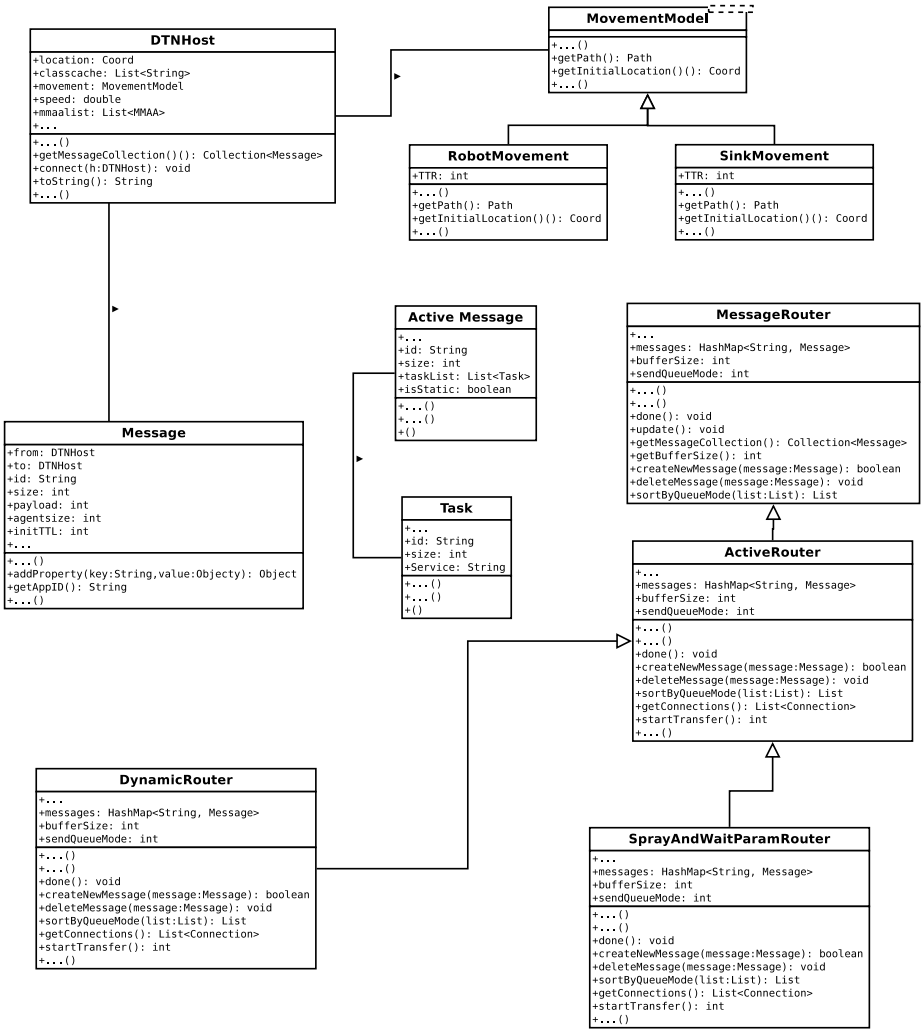


Figure A.1: The ONE dynamic routing, motion report and class caching

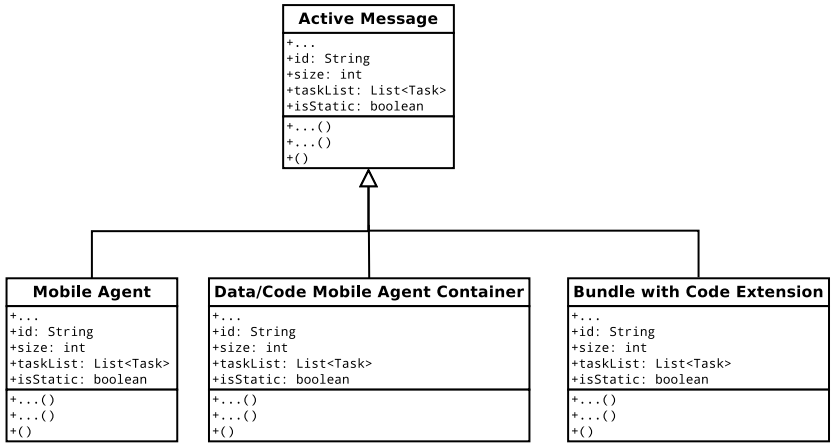


Figure A.2: The three different active message reifications are simulated.

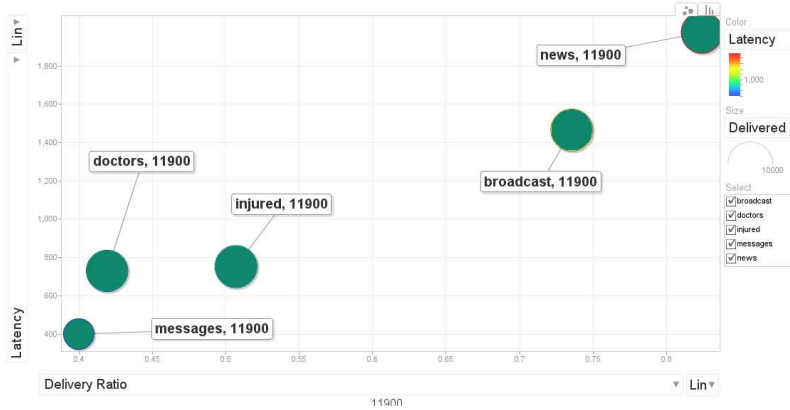


Figure A.3: Motion report

A.3, the applications with *bundle* suffix, their data messages are routed using a single routing protocol. Instead, the applications with *dyn* suffix each employ a different protocol chosen by the application itself. The dynamic approach is significantly better in terms of the number delivered (circle area), latency (x-axis) and delivery ratio (y-axis) for the four applications.

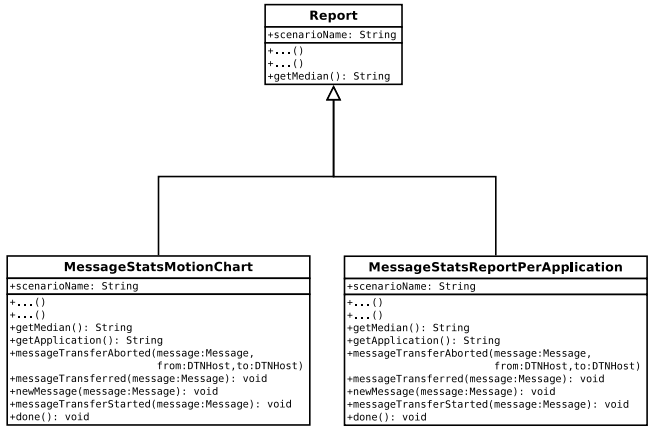


Figure A.4: Report class for The ONE

A.3 Routing Algorithm Parameters Optimization Model

As introduced in Section 8.3, internal optimal DTN routing algorithms parameters or application creation network variables, may change in terms of the context, from application to application, and from DTN node to DTN node. In order to implement several context-aware, application-based dynamic configuration procedures to improve DTN routing protocols and application creation decisions, we need first to establish which are the routing parameters and application creation network variables that will be modified.

Redundancy using codes like Reed-Solomon may be applied while sending copies of application messages in opportunistic networks. When the contact windows are very small, sending fractions of code-message-blocks may perform better. A redundancy factor k is chosen in order to allow s sized messages fragmentation into n, s/m sized blocks ($k = n/m, k \geq 1$). Reconstruction is performed at the destination node from any m different fragments. On the other hand, each fragment can be originally replicated using a factor L, as described in routing algorithms such as Spray and Wait [30]. In Figure A.5, message redundancy and replication is depicted. The different fragments travel independently from node to node heading to their destination where information is

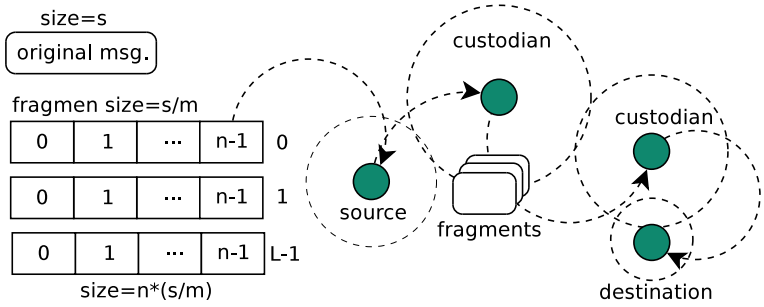


Figure A.5: Replication and redundancy

de-fragmented or eventually discarded. We have developed a The ONE extension which allows us to study the effect of modifying on the fly the variables k, L, s factors to improve latency times and delivery ratio. In future research studies the results will be presented.

Part VI
Bibliography

Bibliography

[1] G. Aad et. al. *The ATLAS Experiment at the CERN Large Hadron Collider.* Journal of Instrumentation, Volume 3, August 2008.

[2] I.F. Akyildiz, Y. Sankarasubramaniam, E. Cayirci. *Wireless sensor networks: a survey.* Computer Networks Volume 38, Issue 4, 15 March 2002, Pages 393-422, 2002.

[3] H. M. Almasaeid, A. E. Kamal. *Data delivery in fragmented wireless sensor networks using mobile agents.* Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems China, Crete Island, Greece: ACM, 2007.

[4] J. Andreeva, C. Borrego, et al. *Automating ATLAS Computing Operations using the Site Status Board.* Computing in High Energy and Nuclear Physics 2012, New York, NY, USA, 21 - 25 May, 2012.

[5] N. Asokan, K. Kostiainen, P. Ginzboorg, J. Ott, C. Luo. *Applicability of identity-based cryptography for disruption-tolerant networking.* Workshop on Mobile Opportunistic Networks, MobiOpp 2007, 2007.

[6] F. Bellifemine, G. Rimassa, A. Poggi. *JADE - A FIPA-compliant Agent Framework.* Proceedings of the 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents, London, 1999.

[7] M. Blanchet. *Delay-Tolerant Networking Bundle Protocol IANA Registries.* RFC 6255, http://tools.ietf.org/html/rfc6255, May 2011.

[8] D. Boneh, M. Franklin. *Identity-Based Encryption from the Weil Pairing.* SIAM J. Comput., Volume 32, Number 3, Pages 586-615, March 2003.

[9] T. Berners-Lee, R. Fielding, L. Masinter. *RFC 3986. Uniform Resource Identifier (URI): Generic Syntax..* RFC 3986, http://www.ietf.org/rfc/rfc3986.txt, January 2005.

[10] C. Borrego. *Publicación de Información y Monitorización relativa usando Agentes Móviles en la Computación Grid.* Master Thesis Research Study, DOI: 10.1109/TIT.2011.2119465, 2008.

[11] C. Borrego, S. Robles. *A store-carry-process-and-forward Paradigm for Intelligent Sensor Grids.* Journal of Information Sciences, DOI information: 10.1016/j.ins.2012.08.016, 2013.

[12] C. Borrego, S. Robles. *Relative Information in Grid Information Service and Grid Monitoring Using Mobile Agents.* 7th International Conference on Practical Applications of Agents and Multi-Agent Systems PAAMS 2009, 2009.

[13] C. Borrego et al. for ATLAS collaboration. *ATLAS Site Status Board. Automatic exclusion and a monitoring on ATLAS computing activities.* Iberian Grid Infrastructure Conference IBERGRID 2011, 2011.

[14] C. Borrego, S. Robles. *Seguridad en la planificación de agentes móviles en redes DTN.* Reunión Española sobre Criptología y Seguridad de la Información, RECSI 2010, 2010.

[15] C. Borrego, S. Robles. *Mobile Agent Virtual Organisation to Improve Relative Information in Grid Services.* Proceeding 3PGCIC '10 Proceedings of the 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2010.

[16] N. Bulusu, D. Estrin, L. Girod, J. Heidemann. *Scalable coordination for wireless sensor networks: self-configuring.* International Symposium on Communication Theory and Applications (ISCTA 2001), Ambleside, UK, July 2001.

[17] G. Burdman, M. Perelstein, A. Pierce. *Large Hadron Collider Tests of the Little Higgs Model.* Journals of Phys. Rev. Lett., Volume 90, Issue 24, 2003.

[18] S. Burleigh. *Delay-Tolerant Networking Architecture.* RFC 4838, http://tools.ietf.org/html/rfc4838, April 2007.

[19] S. Burleigh. *Bundle Protocol Extended Class Of Service (ECOS).* Internet-Draft, http://tools.ietf.org/html/draft-irtf-dtnrg-ecos-03, October 2010.

[20] S. Burleigh. *Licklider Transmission Protocol - Specification.* RFC 5326, http://tools.ietf.org/html/rfc5326, September 2008.

[21] S. Burleigh, E. Jennings, J. Schoolcraft. *Autonomous Congestion Control in Delay-Tolerant Networks.* American Institute of Aeronautics and Astronautics, 2007.

[22] B. Burns, O. Brock, B.N. Levine. *Autonomous enhancement of disruption tolerant networks.* International conference on Robotics and Automation, ICRA 2006, 2006.

[23] Y.U. Cao, A.S. Fukunaga, A.B. Kahng. *Cooperative Mobile Robotics: Antecedents and Directions.* Journal of Autonomous Robots, Volume 4, Pages 226-234, 1997.

[24] V. Cahill, S. Farrell, J. Ott. *Special issue of computer communications on delay and disruption tolerant networking.* Computer Communications Volume 32, Issue 16, 15 October 2009, Pages 1685-1686, 2009.

[25] A. Carzaniga, G.P. Picco, G. Vigna. *Is Code Still Moving Around? Looking Back at a Decade of Code Mobility.* Companion to the proceedings of the 29th International Conference on Software Engineering, 2007.

[26] B. Chen, H.H. Cheng, J. Palen. *Mobile-C: a mobile agent platform for mobile C-C++ agents.* Software—Practice & Experience archive, Volume 36 Issue 15, December 2006.

[27] B. Chen, H.H. Cheng, J. Palen. *Agent-Based Real-Time Computing and Its Applications in Traffic Detection and Management Systems.* ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2004), 2004.

[28] B. Chen, W. Liu. *Mobile Agent Computing Paradigm for Building a Flexible Structural Health Monitoring Sensor Network.* Computer-Aided Civil and Infrastructure Engineering, Volume 25, Issue 7, Pages 504–516, October 2010.

[29] A. Chen, Y. Tang, Y. Liu, Y. Li. *MAGMS: Mobile Agent-Based Grid Monitoring System.* Lecture Notes in Computer Science, ISSN 0302-9743, 2006.

[30] H. Che-Jung, L. Huey-Ing, S. Winston. *Opportunistic routing - A review and the challenges ahead.* Computer Networks, Volume 55, Issue 15, Pages 3592-3603, DOI 10.1016/j.comnet.2011.06.021, 2011.

[31] J. Cucurull. *Efficient Mobility and Interoperability of Software Agents.* Phdthesis, Universitat Autònoma de Barcelona, 2008.

[32] J. Cucurull, R. Martí, G. Navarro-Arribas, S. Robles, J. Borrell. *Agent mobility architecture based on IEEE-FIPA standards.* Computer Communications Volume 32, Issue 4, March 2009.

[33] J. Cucurull, G. Navarro-Arribas, R. Martí, G. Navarro-Arribas, S. Robles, J. Borrell. *An efficient and secure agent code distribution service.* Software: Practice and Experience, Volume 40, Issue 4, Pages 363–386, April 2010.

[34] K Czajkowski, S Fitzgerald, I Foster, C Kesselman. *Grid Information Services for Distributed Resource Sharing.* 10th IEEE International Symposium on High Performance 2001, Pages 181-194, 2001.

[35] F. De Rango, M. Tropea; G. B. Laratta, S. Marano. *Hop-by-Hop Local Flow Control over InterPlaNetary Networks based on DTN.* Architecture,¿ IEEE ICC 2008, Bejing, China, May 2008.

[36] S. Dimitriou, V. Tsaoussidis. *Effective Buffer and Storage Management in DTN Nodes.* International Conference on Ultra Modern Telecommunications and Workshops, 2009.

[37] N. Dimokas, D. Katsaros, L. Tassiulas, Y. Manolopoulos. *High performance, low complexity cooperative caching for wireless sensor networks.* Journal Wireless Networks, Volume 17 Issue 3, April 2011.

[38] G. Dong, W. Tong. *A Mobile Agent-based Grid Monitor Architecture.* Computer Systems and Applications, AICCSA'07. IEEE/ACS, 2007.

[39] L.L. Errol, X. Guoliang. *Relay Node Placement in Wireless Sensor Networks.* IEEE Transactions on Computers, Volume 56, Pages 134-138, 2007.

[40] X. Espinal, C. Borrego, et al. for ATLAS collaboration. *Distributed ATLAS computing activities.* IBERIA Proceedings of the 2nd Iberian Grid Infrastructure Conference, Pages 19-30 Porto, Portugal, May 12-14, 2008.

[41] S. Farrell, V. Cahill. *Delay-and Disruption-Tolerant Networking.* Artech House, Inc. Norwood, MA, USA, 2006.

[42] J. Flix, C. Borrego, et al. *File Transfer Service and CMS data transfer optimizations at PIC Tier-1 center.* EGEE 2007 Conference, Budapest, 2007.

[43] I. Foster, C. Kesselman, S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of High Performance Computing Applications, Vol. 15, No. 3, Pages 200-222, 2001.

[44] G. Fox Dennis, D. Gannon , M. Thomas. *A Summary of Grid Computing Environments.* Concurrency and Computation: Practice and Experience, Special Issue Grid Computing Environments, Volume 14, Issue 13-15, Pages 1035–1044, December 2002.

[45] C. Garrigues, S. Robles, J. Borrell. *Securing dynamic itineraries for mobile agent applications.* Journal of Network and Computer Applications, Volume 31, Issue 4, Pages 487-508, November 2008.

[46] W. Guiyi, L. Yun, V. Athanasios et al. *PIVOT: An adaptive information discovery framework for computational grids.* Information Sciences. Volume 180 Issue 23, December 2010.

[47] B. Hashii, S. Malabarba, R. Pandey et al. *Supporting reconfigurable security policies for mobile programs.* Computer Networks, Volume 33, Issue 1-6, June 2000.

[48] R.L. Henderson. *Job Scheduling Under the Portable Batch System.* IPPS '95 Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, 1995.

[49] D. Henriksson. *A Caching-Based Approach to Routing in Delay-Tolerant Networks.* Proceedings of 16th International Conference on Computer Communications and Networks, ICCCN 2007, 2007.

[50] F. Hess. *Efficient Identity Based Signature Schemes Based on Pairings.* Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography, ser. SAC '02. London, UK, Springer-Verlag, 2003.

[51] G. Hoblos, M. Staroswiecki. *Optimal design of fault tolerant sensor networks.* IEEE International Conference on Control Applications, Anchorage, AK, September 2000, Pages 467–472, 2000.

[52] Z.P.D. Hossein, C. Schlegel, M.H. MacGregor. *Distributed optimal dynamic base station positioning in wireless sensor networks.* Computer Networks, Volume 56, Issue 1, January 2012.

[53] S. Jain, K. Fall, R. Patra. *Routing in a delay tolerant network.* Applications, Technologies, Architectures, and Protocols for Computer Communication: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications; 30 Aug.-03 Sept, 2004.

[54] J-R. Jiang, Y-L. Lai, F-C. Deng. *Mobile Robot Coordination and navigation with directional antennas in positionless Wireless Sensor Networks.* International Journal of Ad Hoc and Ubiquitous Computing, Volume 7, Number 4, 2011.

[55] A. Keränen, J. Ott, T. Kärkkäinen. *The ONE simulator for DTN protocol evaluation.* Proceeding Simutools '09 Proceedings of the 2nd International Conference on Simulation Tools and Techniques, 2009.

[56] S. Kremer, O. Markowitch, J. Zhou. *An Intensive Survey of Fair Non-Repudiation Protocols.* Computer Communications Volume 25, Issue 17, 2002.

[57] A. Krifa, C. Barakat, T. Spyropoulos. *Message drop and scheduling in DTNs: Theory and practice..* Technical report, HAL INRIA, 2010.

[58] L. Krishnamachari, D. Estrin, S. Wicker. *The impact of data aggregation in wireless sensor networks.* Proceedings of the 22nd Distributed Computing Systems Workshops International Conference, 2002.

[59] H. Kuang, L.F. Bic, M.B. Dillencourt. , *Iterative grid-based computing using mobile agents.* 2002 International Conference on Parallel Processing (ICPP'02), Pages 109-113, 2002.

[60] E. Laure, S.M. Fisher, A. Frohner, C. Grandi, P. Kunszt. *Programming the Grid with gLite.* Computational Methods in Science and Technology, Pages 33-45, 2006.

[61] S. Lee, M. Younis. *Optimized relay placement to federate segments in wireless sensor networks.* IEEE Journal on Selected Areas in Communications, Volume 28, Pages 742-752, June 2010.

[62] L. Lefävre, J.P. Gelas. *Towards interplanetary Grids.* In Workshop on Next Generation Communication Infrastructure for Deep-Space Communications held in conjunction with the Second International Conference on Space Mission Challenges for Information Technology (SMC-IT), Pasadena, California, July 2006.

[63] J. Leitner. *Multi-robot Cooperation in Space: A Survey.* Proceeding AT-EQUAL '09 Proceedings of the 2009 Advanced Technologies for Enhanced Quality of Life, 2009.

[64] Q. Li, D. Rus. *Communication in disconnected ad hoc networks using message relay.* Journal of Parallel and Distributed Computing Volume 63, Issue 1, Pages 75-86, January 2003.

[65] F. Li, N. Seddigh, B. Nandy, D. Matute. *An Empirical Study of Today's Internet Traffic for Differentiated Services IP QoS.* Proceedings of the Fifth IEEE Symposium on Computers and Communications, 2000.

[66] H.B. Lim, Y.M. Teo, P. Mukherjee, V.T. Lam et al. *Sensor Grid: Integration of Wireless Sensor Networks and the Grid.* Proc. of the IEEE Conf. on Local Computer Networks, 2005.

[67] A. Lindgren, K.S. Phanse. *Evaluation of Queueing Policies and Forwarding Strategies for Routing in Intermittently Connected Networks.* First International Conference on Communication System Software and Middleware, 2006.

[68] S.K. Madria, M. Mohania, S.S. Bhowmick et al. *Mobile data and transaction management.* Information Sciences, Volume 141, Issue 3-4, Pages 279-309, April 2002.

[69] R. Martí, S. Robles, A. Martín-Campillo, J. Cucurull. *Providing early resource allocation during emergencies: The mobile triage tag.* Journal of

Network and Computer Applications Volume 32, Issue 6, Pages 1167-1182, November 2009.

[70] A. Martín-Campillo, C. Martínez-García, J. Cucurull, R. Martí, S. Robles, J. Borrell. *Mobile Agents in Healthcare, a Distributed Intelligence Approach.* Studies in Computational Intelligence, Volume 309/2010, 49-80, 2010.

[71] K. McCloghrie, D. Perkins, J. Schoenwaelder. *Structure of Management Information Version 2 (SMIv2).* RFC 2578, http://tools.ietf.org/html/rfc2578, April 1999.

[72] J. Ott, E. Hyytia, P. Lassila, T. Vaegs, J. Kangasharju. *Floating content: Information sharing in urban areas.* IEEE International Conference on Pervasive Computing and Communications (PerCom), 2011.

[73] E. Pignaton de Freitas. *Cooperative Context-Aware Setup and Performance of Surveillance Missions Using Static and Mobile Wireless Sensor Networks.* Halmstad University Dissertations, ISBN-978-91-87045-00-4, 2011.

[74] J. Polastre, J. Hill, D. Culler. *Versatile low power media access for wireless sensor networks.* Proceeding SenSys '04 Proceedings of the 2nd international conference on Embedded networked sensor systems, 2004.

[75] I. Psaras, N. Wang, R. Tafazolli. *Six years since first DTN papers. Is there a clear target?* 1st Extreme Workshop on Communication (ExtremeCom2009), Laponia, Sweden, 2009.

[76] A. Puliafito, O. Tomarchio. *Using mobile agents to implement flexible network management strategies.* Computer Communications, Volume 23, Issue 8, 1 April 2000, Pages 708-719, 2000.

[77] A. Ranganathan, K.A. Berman. *Dynamic state-based routing for load balancing and efficient data gathering in wireless sensor networks.* 2010 International Symposium on Collaborative Technologies and Systems (CTS), 2010.

[78] S. Robles, J. Borrell, J. Bigham, L. Tokarchuk. *Design of a trust model for a secure multi-agent marketplace.* Proceedings of the fifth international conference on Autonomous agents, 2001.

[79] L.H. Sahasrabuddhe, B. Mukherjee. *Multicast routing algorithms and protocols: a tutorial.* Network, IEEE, Volume 14 , Issue 1, Pages 90-102, 2000.

[80] M. Saleem, G. Di Caro, M. Farooq, M. Muddassar. *Swarm intelligence based routing protocol for wireless sensor networks: Survey and future directions.* Information Sciences, Volume 181, Issue 20, Pages 4597-4624, DOI: 10.1016/j.ins.2010.07.005, October 2011.

[81] A. Sánchez, C. Borrego, S. Robles, J. Andújar. *Access control for pro-active messages in DTN networks.* Proceedings for the XII Reunión Española sobre Criptología y Seguridad de la Información, 2012.

[82] G. Sandulescu, S. Nadjm-Tehrani. *Optimising Replication versus Redundancy in Window-aware Opportunistic Routing.* Proceedings of International Conference on Communication Theory, Reliability, and Quality of Service, (part of NexCOMM 2010), IEEE, June 2010.

[83] J. Scott, P. Hui, J. Crowcroft, C. Diot. *Haggle: A Networking Architecture Designed around Mobile Users.* Proceedings for the Third IFIP Wireless on Demand Network Systems Conference, 2006.

[84] K. Scott, S. Burleigh. *Bundle Protocol Specification.* RFC 5050, http://tools.ietf.org/html/rfc5050, November 2007.

[85] M. Seligman, K. Fall, P. Mundur. *Storage routing for dtn congestion control.* Wireless Communications & Mobile Computing, Volume 7, Issue 10, December 2007.

[86] A. Seth, S. Keshav. *Practical Security for Disconnected Nodes.* Proceedings of the 1st IEEE ICNP Workshop on Secure Network Protocols, 2005.

[87] T.K. Shih. *Mobile agent evolution computing.* Information Sciences, Volume 137, Issues 1–4, Pages 53–73, September 2001.

[88] R. Sugihara, R.K. Gupta. *Optimal Speed Control of Mobile Node for Data Collection in Sensor Networks.* Sensor Networks, IEEE Transactions on Mobile Computing, Volume 9, Number 1, Pages 127-139, January 2010.

[89] S. Symington. *Delay-Tolerant Networking Metadata Extension Block.* Internet Research Task Force (IRTF), Request for Comments: 6258, The MITRE Corporation, Category: Experimental, May 2011.

[90] C.K. Tham, R. Buyya. *SensorGrid: Integrating sensor networks and grid computing.* Techical Report, National University of Singapore, 2003.

[91] O. Tomarchio. *Active Monitoring In Grid Environments Using Mobile Agent Technology.* 2nd Workshop on Active Middleware Services (AMS'00) in HPDC-9, 2000.

[92] S. Venugopal, R. Buyya, L. Winton. *A Grid service broker for scheduling e-Science applications on global data Grids.* Concurrency and Computation: Practice and Experience, Special Issue Middleware for Grid Computing, Volume 18, Issue 6, Pages 685–699, May 2006.

[93] M. Venkataramana, M. Chatterjeea, K. Kwia. *A dynamic reconfigurable routing framework for wireless sensor networks.* Ad Hoc Networks Volume 9, Issue 7, Pages 1270-1286, 2011.

[94] M. Vidal. *A biological atlas of functional maps.* Cell 2001, Volume 104, Pages 333-339, 2001.

[95] M.A.M. Vieira, M.E. Taylor, P. Tandon, M. Jain, R. Govindan, G.S. Sukhatme, M.Tambe. *Mitigating multi-path fading in a mobile mesh network.* Ad Hoc Networks, DOI 10.1016/j.adhoc.2011.01.014, 2011.

[96] D. Waitzman. *IP over Avian Carriers with Quality of Service..* RFC 2549, https://tools.ietf.org/rfc/rfc2549.txt, April 1999.

[97] J. E. White. *Mobile agents make a network an open platform for third-partydevelopers.* Computer 1994, Volume 27, Issue 11, Pages 89-90, ISSN 0018-9162, 1994.

[98] N.J.E. Wijngaards, B.J. Overeinder, M. van Steen. *Supporting internet-scale multi-agent systems.* Data Knowledge Engineering Volume 41, Pages 229-245, 2002.

[99] M. Wooldridge, N.R. Jennings. *Agent Theories, Architectures and Languages: A Survey.* Wooldridge and Jennings Eds., Intelligent Agents, Berlin:Springer-Verlag, 1-22, 1994.

[100] P. Yang, M. Chuah. *Performance evaluations of data-centric information retrieval schemes for DTNs.* Computer Networks, Volume 53, Issue 4, Pages 541-555, DOI 10.1016/j.comnet.2008.09.023, 2009.

[101] Y. Zhang, W. Liu, Y. Fang, D. Wu. *Secure Localization and Authentication in Ultra-Wideband Sensor Networks.* IEEE J. Selected Areas in Comm., Volume 24, Number 4, Pages 829-835, April 2006.

[102] S. Zhanfeng, L. Jiancheng, H. Guangyu et al. *Distributed computing model for processing remotely sensed images based on grid computing.* Information Sciences, Volume 177, Issue 2, Pages 504-518, DOI 10.1016/j.ins.2006.08.020, 2007.

[103] Z. Zhang. *Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges.* IEEE Communications Surveys Tutorials (2006) Volume 8, Issue 1, Pages 24-37, 2006.

[104] H. Zhu, X. Lin, R. Lu, Y. Fan, X. Shen. *SMART: A Secure Multilayer Credit-Based Incentive Scheme for Delay-Tolerant Networks.* IEEE Transactions on Vehicular Technology, Volume 58 , Issue 8, 2009.

[105] N. Zimmerman, J. Ramos, R. Salguero-Gomez. *The next generation of peer reviewing.* Frontiers in Ecology and the Environment, Volume 9, Issue 4, May 2011.

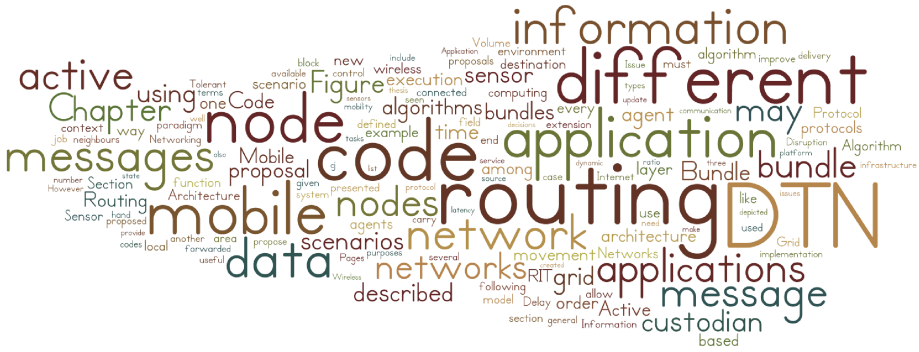
[106] *Ohio University's Off-World Communication Protocol research group Java Bundle Protocol Implementation.* [Online] http://irg.cs.ohiou.edu/ocp/bundling.html - [last access: 3/1/2013].

[107] *DTNRG: Delay Tolerant Networking Research Group.* [Online] http://www.dtnrg.org - [last access: 3/1/2013].

[108] *Networking Laboratory, Helsinki University of Technology.* [Online] http://www.netlab.tkk.fi/engl.shtml - [last access: 3/1/2013].

[109] *SENDA research group.* [Online]. *https://senda.uab.cat - [last access: 3/1/2013].*

Thesis Hash:



Hash Signature:

Carlos Borrego Iglesias
Bellaterra, January 2013