

UNIVERSITÉ DE NEUCHÂTEL
FACULTÉ DES SCIENCES
Institut d'informatique

T H È S E

Soutenue par

Raphaël KUMMER

le 06.09.2010

Information Lookup and Distribution in Mobile Ad hoc Networks

Acceptée sur proposition du jury:

Prof.	Peter	Kropf	<i>Université de Neuchâtel</i>	Directeur de Thèse
Prof.	Pascal	Felber	<i>Université de Neuchâtel</i>	Rapporteur
Prof.	Dimitri	Konstantas	<i>Université de Genève</i>	Rapporteur
Prof.	Rui	Olivera	<i>Universidade do Minho</i>	Rapporteur

IMPRIMATUR POUR LA THESE

Information Lookup and Distribution in Mobile Ad hoc Networks

Raphaël KUMMER

UNIVERSITE DE NEUCHATEL

FACULTE DES SCIENCES

La Faculté des Sciences de l'Université de Neuchâtel,
sur le rapport des membres du jury

MM. P. Kropf (directeur de thèse), P. Felber,
Rui Oliveira (Universidade do Minho, Portugal)
et Dimitri Konstantas (Université de Genève)

autorise l'impression de la présente thèse.

Neuchâtel, le 23 septembre 2010

Le doyen :
P. Kropf

P. Kropf

Acknowledgments

During my thesis I met many people. It will be impossible to cite them all here, but I would like to thank them all for their valuable inputs, remarks and ideas during these years.

I would like to thank particularly Olena Zubaryeva for her constant and unconditional support. Her advices greatly helped me to go forward and achieve this work, in particular during the last year.

People who also played an important role during these years are all my colleagues and friends from the computer science department of the university of Neuchâtel, namely Marc, Christian Spielvogel, Sabina, Leo, Fred, Steve, Silvia, Heiko, Claire, Lucas, Derin, Cristian Gatu, Patrick, Ljiljana and Walther. I had great time with them during and outside work.

I'm very thankful to Christian, Derin and Lucas who greatly helped me to prepare the thesis exam.

I want to thank to Professor Peter Kropf, my advisor, for his help to go forward during these years. Also many thanks to Professor Pascal Felber for his support and helpful advice during my thesis work.

Thanks to Professor Rui Olivera and Dimitri Konstantas for being part of the Jury.

My parents have been the architects of this work by supporting me all along my studies. They provided me with everything necessary to advance in my life, studies and work. I'm grateful for their support.

Finally, I would also like to express my gratitude to everyone who has been present during all these years but not cited here.

Abstract:

In this thesis, we propose two novel algorithms, a lightweight distributed hash table (DHT) lookup algorithm and a multicast tree building algorithm, specifically designed to take advantages of the major characteristics of mobile ad-hoc networks (MANETs) like broadcast communication, limited resources (particularly energy) physical proximity or multi-hop communication that are often put aside or considered as restrictive factors in proposed information lookup and distribution solutions.

Our DHT lookup algorithm is adapted to MANETs specificities. The physical proximity of nodes, the broadcast communication and the message relaying process used to route the messages are exploited to find shortcuts in the DHT logical space. These shortcuts are then used to quickly converge to the node responsible for the searched data(i.e., key in the logical space).

The multicast tree building algorithm operates on top of the lightweight DHT. It builds efficient multicast trees in MANETs that strives to limit the number of relay nodes and the number of required transmissions. The so build content distribution structure fits the physical network topology and adapts when node are moving.

The algorithmic efficiency and scalability are evaluated by the means of simulations with various network sizes and configurations. The obtained results demonstrate that both algorithms behave well in MANETs

Finally, we detail the Adcast software implementing our both algorithms. It runs on Windows mobile powered PDAs and Windows computers. Adcast allows to evaluate the algorithms in real and simulated ad-hoc networks.

Keywords: Wireless Mobile Ad-hoc Networks, Distributed Hash Tables, Multicast, Distributed Systems, Mobile Technologies, Networking Technologies, Algorithms, Wireless Technologies, Middleware, Windows mobile.

Résumé:

Dans cette thèse, nous proposons deux nouveaux algorithmes: un algorithme de recherche pour table de hachage distribuée (DHT) et un algorithme construisant des arbres de multicast. Ces algorithmes ont été spécifiquement conçus pour tirer avantage des caractéristiques principales des réseaux mobiles ad-hoc (MANETs) telles que la communication par diffusion, les ressources limitées (en particulier l'énergie), la proximité physique ou encore les communications nécessitant plusieurs sauts. Ces particularités ne sont généralement pas prises en compte ou considérées comme des facteurs limitatifs dans les solutions conçues pour la recherche et la distribution d'informations dans les MANETs.

Notre algorithme de recherche pour DHT est adapté aux spécificités des MANETs. La proximité physique des noeuds, la communication par diffusion et le mécanisme de routage des messages sont utilisés pour trouver des raccourcis dans l'espace logique de la DHT. Ces raccourcis permettent une convergence rapide vers le noeud responsable pour les données recherchées (une clé dans l'espace logique.)

L'algorithme de multicast fonctionne au-dessus de la DHT. Il construit des arbres de multicast dans les MANETs qui limitent le nombre de relais et de transmissions. La structure ainsi générée est ajustée à la topologie physique du réseau et s'adapte en continu lorsque les noeuds se déplacent.

L'efficacité des algorithmes ainsi que leur capacité de passage à l'échelle ont été évaluées au moyen de simulations dans des réseaux de tailles et de configurations diverses.

Finalement, nous présentons Adcast, le logiciel où sont implémentés nos algorithmes. Il s'exécute sur des PDAs fonctionnant sous Windows Mobile et des ordinateurs sous Windows. Adcast a rendu possible des tests dans des environnements réseaux ad hoc réels et simulés.

Mots-clés: Réseaux sans fil mobiles ad hoc, Table de hachage distribuée, Multicast, Système distribué, Technologies mobiles, Technologies réseaux, Algorithmes, Technologies sans fil, Middleware, Windows mobile.

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivations	2
1.3	Research objectives	4
1.4	Contributions	6
1.4.1	Distributed hashtable for mobile ad hoc networks	6
1.4.2	Multicast tree building algorithm for mobile ad hoc networks	7
1.5	Evaluation methodology	7
1.6	Thesis organization	9
2	Distributed hash table for MANET	11
2.1	Introduction	11
2.2	Related work	12
2.3	General concept	17
2.4	The basic algorithm	19
2.5	Convergence and termination	21
2.6	Increasing visibility	22
2.7	Exploiting request history	23
2.8	Evaluation methodology	24
2.8.1	Experimental simulation environment	24
2.8.2	Experimental simulation results	28
2.8.3	Experimental emulation environment	35
2.8.4	Experimental emulation results	38
2.9	Summary and discussion	42
3	Building multicast trees in MANETs	45
3.1	Introduction	45
3.2	Related work	46

3.3	Design rationale	50
3.4	General concepts	53
3.5	The basic algorithm	55
3.6	Finding more potential parents	56
3.7	Finding better parents	57
3.8	Removing redundant parents	59
3.9	Overhearing multicast messages	61
3.10	Evaluation methodology	63
3.10.1	Experimental environment	63
3.11	Experimental results	67
3.11.1	Degree of member nodes	68
3.11.2	Relay nodes	69
3.11.3	Average distance between member nodes	71
3.11.4	Transmissions required for reaching all members	72
3.11.5	Multicast cost	72
3.11.6	Scalability	74
3.11.7	Variability of the results	75
3.11.8	Structure of the tree	75
3.12	Summary and Discussion	77
4	The Adcast software	79
4.1	Introduction	79
4.2	Software architecture and features	81
4.2.1	Sending and receiving layer	84
4.2.2	Routing layer	85
4.2.3	Lookup layer	87
4.2.4	Multicast layer	88
4.3	Algorithms	90
4.3.1	The AODV implementation	90
4.3.2	The DHT implementation	93
4.3.3	The multicast tree building algorithm implementation	96
4.4	Experimentations	98

Contents	11
4.4.1 Experimental setup	98
4.4.2 Experimental results	101
4.5 Summary and discussion	106
5 Conclusion	109
5.1 Contributions	110
5.1.1 Distributed hashtable for MANETs	110
5.1.2 Multicast tree building algorithm for MANETs	112
5.2 Discussion	113
5.3 Future work	115
A Freemote	119
A.1 Introduction	119
A.2 P2P mobile ad-hoc system testbeds	122
A.3 Wireless ad-hoc network emulation environment	124
A.3.1 Freemote emulator	126
A.3.2 Software implementation	129
A.3.3 MANET simulations	133
A.4 Conclusion	136
Bibliography	137

List of Figures

2.1	Illustration of the DHT model for ad-hoc networks	19
2.2	Comparison of the number of logical hops started end terminated	28
2.3	Percentage of terminated logical hops	30
2.4	Average number of physical steps to achieve a request	31
2.5	Average number of logical hops per lookup	32
2.6	Average number of logical hops per lookup in different network sizes	33
2.7	Path of a request routed by the DHT with all extensions	33
2.8	Stretch of a DHT request vs direct route	35
2.9	Average use of logical shortcuts during a request's logical path . .	39
2.10	Average number of physical steps to achieve a request	40
3.1	Illustration of the utility of our algorithm.	51
3.2	Sample tree illustrating the used terminology.	54
3.3	Sample scenarios illustrating a tree not organized optimally. . . .	58
3.4	Scenarios of Figure 3.3 after applying optimizations.	60
3.5	Cumulative percentage of members acting as internal members as a function of their degree (static - mobile comparison).	68
3.6	Percentage of relay nodes involved (static - mobile comparison). .	70
3.7	Average number of physical steps separating two members (static - mobile comparison).	71
3.8	Average number of transmissions required to cover the tree (static - mobile comparison).	73
3.9	Average multicast cost ($R(n)$) (static mobile comparison).	73
3.10	Multicast cost evolution in different network sizes (static configu- ration).	74
3.11	Sample tree built by the basic algorithm (static configuration). . .	76
3.12	Sample tree built by the improved algorithm (static configuration.)	76
4.1	Adcast start window (i.e., general configuration)	82

4.2	Adcast routing window	82
4.3	Adcast DHT window	82
4.4	Adcast multicast window	82
4.5	Layers structure and roles	83
4.6	Interactions between layers	86
4.7	Experimental network's simulated topology	99
4.8	Experimental multicast tree #1 after connection	103
4.9	Experimental multicast tree #1 at the end of the experiment . . .	103
4.10	Experimental multicast tree #2 after connection	104
4.11	Experimental multicast tree #2 at the end of the experiment . . .	105
A.1	Typical deployment during an Experiment	127
A.2	GUI with emulated node visualization tool	128
A.3	Layered view of software architecture	130
A.4	Layered view of software architecture for "ping" application illus- trated in Figure A.6	131
A.5	Code templates used to facilitate implementation of each layer . .	131
A.6	Freemote experimental "ping" configuration mixing real and emu- lated nodes	132

List of Tables

2.1	Average use of logical shortcuts	29
2.2	Average cost of a logical hop in terms of physical steps	31
2.3	Average percentage of logical shortcuts $N/A =$ Not Available . . .	38
2.4	Average number of physical steps per request	41
2.5	Average gain in physical steps compared to “Basic” DHT, for NoN and Cache DHT	41
3.1	Terminology	53
3.2	Percentage of relay nodes involved in multicast trees with direct request or DHT lookup.	67
4.1	Average use of logical shortcuts	102
A.1	Data link and physical layer features	134

Introduction

1.1 Context

Wireless technologies have become ubiquitous. They provide global (i.e., everywhere) connectivity in urban areas and allow connecting off-centered areas to information networks. An ever growing number of individuals now uses them by the means of portable phones or portable computers in order to surf the web. A large number of applications for wireless devices have become popular with the drastic increase of the number of access points providing wireless service.

However, phone calls and web surfing are not the only applications for wireless enabled devices. They may also form ad-hoc networks where no specific infrastructure like access points is required, thus producing a self-organized network where each node is able to directly communicate with its close physical neighbors.

Despite the different kinds of involved devices and communication standards, the constraints remain the same. An ad-hoc network is most of the time composed of small, simple, battery powered devices with limited resources (i.e., memory and CPU) that can be deployed and relocated everywhere. Communication between remote nodes usually requires multiple hops via relay nodes since a node can communicate directly only with its physical neighbors (i.e., the nodes located in its communication range). Thus each communication has a non-negligible cost leading to a global network lifetime reduction.

As new devices with ad-hoc networking capacities and growing but still limited resources are developed, their use has been extended from their known typical application, such as wide area monitoring, to new types of applications involving information lookup and distribution that require novel and efficient solutions. The possible applications outside monitoring are numerous, including, but not limited to file exchange, information services and communication services (i.e., instant messaging).

This work focuses on efficient lookup and multicast mechanisms to efficiently locate and distribute information in mobile ad-hoc networks (MANETs).

1.2 Motivations

Centralized approaches based on the client-server model cannot achieve acceptable performances in mobile ad-hoc networks. Indeed, none of the participating nodes has typically enough capacities nor connection stability to support a server service. Generally, frequent disconnections (churn) and topology changes are prohibitive for such a model in MANETs.

In fact, as nodes may often leave and join the network and because of the mobility and limited resources (disconnecting because of lack of energy, moving outside of the connectivity area) no single node will be capable to reliably support a scalable server.

As the nodes constituting a MANET have only limited energy, bandwidth, memory and processing capacities, a suitable system must rely on the cooperation between the nodes. The Peer-to-Peer (P2P) paradigm where all nodes participate in the system operations offers such a collaborative approach. In this context, two main approaches have emerged over the standard Internet network infrastructure: structured and unstructured algorithms.

Unstructured approaches for P2P file sharing like Gnutella or KaZaA [Gnu 2008, Kaz 2008] have typically neither control on the topology nor on the file placement, thus they often simply flood the network to locate data, with a high risk to overload the network. Unstructured approaches are therefore not adapted for MANETs since they scale poorly and they need many transmissions and much energy to locate the desired content.

Structured solutions use specialized placement algorithms to assign responsibility of each content (file) to a specific node. They use directed search protocols to efficiently locate the files with few communications and are thus better suited for MANET environments. An important class of structured solutions is based on distributed hash tables (DHT) where each item and node is identified by a unique key. Nodes are placed in that logical space accordingly to their logical key and they are responsible for the item having the smallest logical distance to them. Furthermore, facilities are provided to locate a node responsible for a specific key without flooding and without producing false negative (i.e., the search fails only if there is no matching file in the system).

However, well known DHT solutions like Chord [Stoica 2001], Pastry [Rowstron 2001a] or CAN [Ratnasamy 2001] are not suitable for ad-hoc networks, as they do not consider the physical location of nodes when they create their logical overlay network. Considering this fundamental gap between logical and physical spaces, the ad hoc network will be overloaded by the service messages necessary to maintain the logical neighborhood relations through expensive multiple hop path, thus making a simple mapping from a DHT design to ad-hoc networks unrealistic.

Similarly, flooding-based solutions will not achieve acceptable performances to distribute content to a subset of interested nodes. Indeed, using this kind of approach in MANETs forces all the nodes to participate, even if they are not interested in the distributed content. This leads to ineffective solutions where a lot of energy and bandwidth are consumed to achieve only poor performance as the network gets quickly overloaded. One can also use flooding to build a shortest path spanning tree. Although flooding is reduced, such trees make an inefficient usage of the (limited) resources, as they may include many nodes not interested in the content. Furthermore, some flooding is still required to build the tree, and no efficient solution is provided to the content provider to efficiently locate the multicast group covered by the spanned tree.

Superimposing a structured overlay substrate on top of the physical network, as done for instance by Scribe [Rowstron 2001b], can help to construct more efficient trees for multicasting. Here, multicast trees are rooted at “rendez-vous nodes” managed by the underlying Pastry [Rowstron 2001a] DHT. Nodes interested in joining a multicast group route a request via Pastry towards the source and connect to the first group member reached on the way to the rendez-vous point. While this strategy is effective in wired networks, it cannot easily be transposed to mobile ad-hoc networks where communication is multi-hop and physical proximity is an essential consideration.

1.3 Research objectives

From a general point of view, the operating way of unstructured algorithms is prohibitive for MANETs. They generate many transmissions and thus overload the network. Consequently, the general challenge in the context of MANETs consists here in exploiting the interesting properties of structured overlays (DHT and Multicast trees) identified in wired networks and overcoming the operational problems caused by the environmental particularities. The proposed solutions must reach the following objectives:

Achieve good performance: The properties of structured solutions are interesting as they limit the number of communications (i.e., transmissions) required to locate or distribute information. Consequently, we would like them also to perform well after being adapted for MANETs. The goal here is to provide algorithmic adaptation for the wireless networking environment that will not alter the good performance of these structured solutions.

Consider networking environment particularities: Unlike in the wired Internet, the physical proximity/position is a challenge that cannot be ignored in MANETs. A node can only communicate directly with its physical neighbors and may often change its position (i.e., physical neighborhood). Thus, our goal is to define light overlay structures that require a minimal number of transmissions for maintenance (particularly the DHT) and that easily adapt to frequent topology changes (particularly the multicast tree).

Limit resource consumption (energy, CPU, memory, bandwidth): The devices involved in MANETs are, most of the time, small sized and battery powered and use wireless technologies to communicate. Their computational capacity and available memory are also limited. The challenge is then to propose solutions that require a limited number of transmissions (i.e., spare bandwidth and energy) and that are able to share the work and memory load between the participating nodes. In addition, the number of nodes passively involved in the algorithms' operations should be minimized in order to spare their resources (battery).

1.4 Contributions

In this thesis, we present two distributed algorithms based on the P2P paradigm adapted for MANETs¹. The first one is a Distributed Hashtable (DHT) algorithm and the second, running upon the DHT, is a multicast tree building algorithm. These algorithms are designed to provide efficient services fulfilling our research objectives presented in section 1.3. Both are based on ideas that have demonstrated their efficiency on wired networks but are unsuitable for MANETs as discussed in section 1.2.

The algorithms are designed to take advantage of MANETs specificities rather than being hampered by them. Additionally, we try to limit the indirect implication of nodes to reduce their resource consumption.

1.4.1 Distributed hashtable for mobile ad hoc networks

Our first contribution consists of a DHT lookup algorithm specifically designed for MANETs. It combines a minimalist overlay structure together with an adaptive routing mechanism to quickly locate the content. On one hand, as in the well-known wired approaches like Chord or Pastry, the nodes are organized in a logical ring. On the other hand, no long-range links for logical shortcuts are created as their maintenance cost will be prohibitive. Alternatively, the physical neighborhood of the nodes traversed by requests provides cheap shortcuts in order to quickly converge toward the destination in the logical space. Additionally, we propose extensions in consideration with an extra level of visibility in the physical neighborhood (neighbors of neighbors) and with memorizing previous requests to dynamically identify and exploit possible shortcuts.

¹Some parts of this work have already been published in conference papers and book chapters [Kummer 2006b, Kummer 2006a, Kummer 2008, Seet 2009]

1.4.2 Multicast tree building algorithm for mobile ad hoc networks

The second contribution is an algorithm for building multicast trees in mobile ad-hoc networks. It uses the specialized lightweight DHT overlay. A tree is created by the source (i.e., by the multicast group), which can be efficiently localized by DHT lookup. Several techniques to connect joining nodes to a physically close member are proposed as well as extensions to reduce the number of relay nodes involved in the distribution of messages. These additions also contribute to maintain the tree structure regardless of the nodes movements.

The evaluation methodology used to evaluate these algorithms is described below.

1.5 Evaluation methodology

Simulations have been conducted to evaluate the performance of both algorithms (lookup and multicast) in different scenarios. We did not take into account churn (nodes frequently joining and leaving the system), as we are primarily interested to evaluate the lookup efficiency and structural properties of the produced multicast trees.

The algorithms have been implemented with the use of a discrete step simulator. Each simulation step represents a time interval Δt . At each simulation step, before executing the algorithms operations, the new position of the node (mobility), and the physical neighborhood (i.e., the node's connectivity) are calculated.

Each algorithm is an independent layer that can communicate with all others through the simulator's services. As the DHT and the multicast tree building algorithm are designed as middleware services (i.e., they operate between low layer services), they require a routing layer to operate. Thus, we implemented the well known Ad hoc On Demand Distance Vector (AODV) algorithm [Perkins 1999]² to support the routing of multi-hop messages.

The proposed DHT algorithm measurements (detailed in Section 2.8) achieve two main goals. Firstly, they supply a base line for comparison with algorithms deployed over the wired Internet. Secondly, they realize the operational cost of the DHT algorithm in MANETs.

To demonstrate that the DHT algorithm is suitable for resource-limited devices like motes³, we implemented it for the Freemote ad-hoc networks emulator [Maret 2007] (see Appendix A). The same algorithm implementation ran on emulated and real JNotes [Maret 2005] connected together by the Freemote infrastructure. Beside demonstrating that the DHT can run in a real environment, this experiment allows us to compare the simulation results with real ones.

The measurements used to evaluate the multicast tree building algorithm (detailed in Section 3.10) are divided in two groups. The first group of evaluation criteria gives information on the tree structure and its stability over the time in the mobile network. The second is composed by scale-free (i.e., does not depend on the number of nodes in the MANET) multicast cost measurements. They provide information on the general multicast efficiency of the algorithms.

²We chose this algorithm as it is well known, but it could be replaced by any MANET routing protocol.

³Motes are also sometimes referred to as smart dust. One mote is composed of a small, low powered and cheap computer connected to several sensors and a radio transmitter capable of forming ad hoc networks.

Finally, to demonstrate the feasibility of this approach, the DHT algorithm and the multicast tree building algorithm have been implemented on an ad hoc network of Windows mobile PDAs for which the Adcast software has been developed.

1.6 Thesis organization

The rest of this thesis is organized as follows: Chapter 2 details the Distributed hashtable algorithm. It is followed by Chapter 3 that presents the multicast tree building algorithm. Chapter 4 describes the Adcast software that implements both algorithms. Finally, Chapter 5 concludes this thesis.

Distributed hash table for MANET

2.1 Introduction

In this chapter, we present our distributed hash table (DHT) lookup algorithm for mobile ad hoc networks. We propose a DHT design that combines a minimalist logical overlay structure together with adaptive routing mechanisms, based on “directed search” in order to quickly locate some content. Nodes are organized in a logical ring, similarly to Chord [Stoica 2001] or Pastry [Rowstron 2001a]. Each data item is identified by a key (data identifier). The DHT maps keys to the nodes of the overlay network.

Contrary to the approaches discussed in Section 2.2 we do not use logical long-range neighbors whose maintenance costs would be prohibitive. Instead, we rely on the physical neighbors of the nodes traversed by requests routed through the physical network to quickly converge toward the destination in the logical overlay. We propose additional extensions, where we consider an extra level of visibility in the physical neighborhood (neighbors of neighbors) and maintain a history of previous requests to dynamically identify and exploit possible shortcuts.

We have extensively studied our algorithm by the means of simulations and emulations in static and mobile deployment scenarios. We did not consider churn because it is a very complex problematic. It is indeed important to design such an algorithm adapted to MANETs before addressing this problem. Therefore a good churn handling approach must be adapted to the intrinsic algorithm operation mode. The evaluation results obtained demonstrate that

our approach is indeed efficient and scales well to large peer or node populations. It represents a good alternative to existing solutions based on unstructured P2P networks.

The remainder of this chapter is organized as follows. Section 2.3 presents the general concept of our algorithm. We then detail the basic algorithm in Section 2.4 and demonstrate its convergence and termination in Section 2.5. Sections 2.6 and 2.7 present the extensions of our basic algorithm. We detail the evaluation methodology in Section 2.8 and present the experimental results in Section 2.8.2. Finally, Section 2.9 concludes.

2.2 Related work

Peer-to-peer overlays have emerged from file sharing applications on top of the Internet, leveraging from its routing infrastructure and the intrinsic peer-to-peer property of the IP protocol. In MANETs, the situation is different as a path between nodes may traverse many relays. Furthermore, two nodes are directly connected only if they are physical neighbors (i.e., in the communication range of each other). In particular, the DHT paradigm with its notion of regular topology (often a ring) and the shortcuts (fingers) introduced at the overlay layer makes a direct mapping to ad-hoc networks difficult as discussed in the previous chapter. Various approaches are known from literature to achieve such mappings. They are broadly presented and discussed in the following paragraphs.

[Kevin 2002] proposed CRAP, an adaptation of the Chord Distributed hashtable for ad hoc networks like sensor networks. The algorithm merges spanning trees in the network until the emergence of a single spanning tree covering the whole network. The link in the spanning tree are pairs of directed edges pointing in opposite directions thus it exists at least one cycle though the nodes. Then the Chord address space is distributed on top of the built

spanning tree. When receiving a request, a CRAP node handles the received request if it is responsible for it. Otherwise the node checks if one of its children is responsible for the key and sends the request to this node or to his parent.

While GRACE (Global Replication And Consistency) [Bosneag 2005] has not been specifically designed with ad-hoc networks in mind, it enables mobile collaboration by combining DHT properties with a layered architecture. GRACE also supports mobility in wide-area networks. The different layers or consistency levels are interconnected through “consistency neighbors” that are logically close to each other. Requests are routed along these neighbors. The lookup algorithm of the system is based on Pastry [Rowstron 2001a]. This approach still relies on the standard Internet infrastructure.

Pucha et al. [Pucha 2004b] implement Pastry on top of the MANET routing protocol DSR (Dynamic Source Routing) [Johnson 1996]. Three modifications are proposed as compared to the implementation on the Internet: (1) the node join procedure is modified by expanding the ring search for locating distinguished bootstrap nodes in charge of arrivals; (2) the Pastry ping metric is replaced by a distance metric to reduce network load; and (3) the DSR protocol is modified to inquire about the proximity used in the adapted Pastry routing.

Ekta [Pucha 2004a] and MADPastry [Zahn 2005] integrate the DHT paradigm with ad-hoc network routing. Both approaches introduce the necessary functions at the network routing layer. The principal idea of Ekta is to move the DHT protocol from the overlay level to the network layer of the MANETs: a one-to-one mapping between IP addresses and logical (DHT) node IDs is applied. MADPastry is built on top of the AODV protocol (Ad-hoc on-demand vector routing) [Perkins 1999]. This protocol aims to avoid full broadcasts as much as possible because these are costly in ad-hoc networks when targeted to the entire network. MADPastry creates clusters composed of physically close nodes sharing, at the same time, a common overlay prefix. The nodes in a cluster are

thus physically and logically close. Routing is then based on the logical overlay node IDs.

All the approaches presented so far suffer from their large sizes of the routing tables and the complexity of setting up and managing connections with all the nodes that they contain.

Cell Hash Routing (CHR) [Araujo 2005] is a specialized ad-hoc DHT. CHR uses position information to construct a DHT of clusters instead of organizing individual nodes in the overlay. This approach groups nodes according to their physical location. The routing between the clusters is done by position-based routing with the GPSR [Karp 2000] routing algorithm. A major limitation of this approach is that nodes are not individually addressable, but only via the clusters.

Cramer et al. [Cramer 2005] propose the Chord based Proximity Neighbor Selection strategy (PNS-CHORD). Here, the nodes are connected to their logical successors on the ring and through logical shortcuts to further nodes as usual in Chord. These logical long-range neighbors are chosen according to the physical proximity in the ad-hoc network and are either one or two steps away. As the construction of the routing tables is based on physical proximity, it may happen that the logical path pursued by a request traverses the same node multiple times. This may only be prevented when the nodes keep track of the requests that pass by and subsequently adjust their routing tables in case the same request passes twice, but on different logical shortcuts.

Castro et al. at Microsoft Research (Cambridge) propose Virtual Ring Routing (VRR) [Castro 2006], a DHT solution for MANETs close to the one presented in the next chapter that similarly aims to combine ad-hoc routing and logical DHT-like addressing. Their approach differs from our algorithm in several ways. First, they build and proactively maintain bidirectional routes between nodes. In contrast, our algorithm always tries to find the best route at

each node and sends the request along this path. It is therefore possible to take into account locally profitable situations. Second, VRR also maintains existing routes in a proactive way. Our solution does not maintain existing routes, but previous routing decisions are kept in a cache and can thus be reactivated if appropriate. Finally, VRR nodes use information of the physical paths traversing the nodes for improving the routing. Our experiments, however, have shown that such a caching policy results in small improvements, only. Indeed, the cached entries are effective only when a request's path goes through the node having registered them. In our DHT algorithm, the nodes also capture communications of the nodes in the physical neighborhood. The information acquired this way implicitly includes routing decisions and thus allows avoiding paths to traverse the same physical area twice. It has to be noted that this information is obtained without any further communication overhead, since it may be captured for free due to the characteristics of wireless networks, i.e., radio transmission.

None of the three previous approaches tries to prevent requests for passing multiple times in the same network area. Only VRR uses information from the physical paths traversing the nodes to improve routing but this information can only be used by the nodes that have memorized it.

More recently, Cramer et al. implemented Chord directly in [Cramer 2006] mobile ad hoc networks to evaluate its performances in this environment. Chord has been simulated on a network of 50 nodes that can move in random direction with a speed uniformly distributed between 0 and 2 m/s. Their results demonstrate that the main issue of implementing Chord in Manets is not the overhead to maintain the finger tables but the pessimistic timeout and fail-over strategy. The strategy of limiting the effect of fleeting node populations discards valuable states in Manets.

In [Heer 2006], Heer et al. analyze Chord behavior in mobile ad hoc networks and proposes adaptation to make the algorithm run in this environment.

They propose a solution adapted the MANETs to discover DHT ring present in the node's surrounding and to join the largest one by analyzing the segment of key managed by known nodes. Then they propose a solution to merge a DHT ring to another one where the joining nodes (usually the nodes from the smallest DHT ring) simply join the other DHT and keep their identifier thus minimizing the changes in ring. Finally, Solutions to handle churn and node instability are studied. The authors propose to avoid relying on instable nodes and routes and to delay recovery processes to deal with temporary nodes' disappearance.

Others, like in [Conti 2004] study the adaptation of Pastry [Rowstron 2001a] in MANETs. The proposed solution is based on a layered architecture where the network information is shared in a cooperative way. The network routing is extended to proactively spread service information together with link state packets. Thus the Pastry routing tables can be deduced from the ones build at routing layer.

Baccelli and Schiller proposed [Baccelli 2008] DHT-OLSR. The protocol is build on the idea of cross-fertilization between mobile ad-hoc networks and peer-to-peer networking. The proposed protocol adds an enhanced implementation of MADPastry [Zahn 2006] with OLSR [Clausen 2003]. When a node has to forward a data packet, first it looks up the destination in its OLSR routing tables. If the destination is found there, the data packet is sent to this node. The OLSR routing table contains the nodes in the direct physical neighborhood (within one or two physical steps). When the packet's destination is not in the node's OLSR routing table it then uses the MADPastry algorithm to communicate with remote nodes. The MADPastry differs here from its original implementation by using OLSR routing tables instead of AODV.

2.3 General concept

In a DHT system, each node and key has a specific position in a logical identifier space thus creating a logical overlay above the physical network. The keys are mapped to nodes accordingly to a proximity metric in the logical space. This allows any node to use this DHT substrate to determine the current live node responsible for a given key. For instance, Chord [Stoica 2001] connects each node to its closest neighbors (successor and predecessor) in the identifier space, hence organizing the nodes in a logical ring. These connections always allow traversing the whole ring, albeit at very high cost. They are necessary and sufficient for the safety and reliability of the system. Additionally, each node has a number of long-range neighbors called fingers for efficient lookup and for maintaining liveness properties. They are located at exponentially increasing distance in the logical space. Using these links, a node can quickly reach remote locations: the expected path length of a lookup becomes $O(\log N)$ hops, where N is the number of nodes in the system.

In the following, we assume that (1) the ad-hoc network forms a connected graph and (2) that there is an underlying ad-hoc routing protocol that allows any node to route a message towards any other one. The second assumption stems from the fact that nodes can directly communicate only with their physical neighbors (i.e., the nodes within their communication range). We do not make assumptions on the ad-hoc routing protocol except that it always succeeds and it is possible to interrupt the route of a message at intermediate nodes.

Wired network approaches like Chord [Stoica 2001] or Pastry [Rowstron 2001a] never consider the physical position of nodes. As a result, successors, predecessors and long-range links connect remote nodes through multiple physical steps. Besides being inefficient in terms of physical path length, such approaches are impracticable because each node would have to maintain accurate routing information for $O(\log N)$ long-range neighbors. In case of larger networks and the addition of churn and mobility, this becomes a major problem and leads to

a considerable amount of traffic, hence overloading the network. Thus, a direct mapping of a wired DHT design to MANETs is unrealistic.

Therefore, the ad-hoc DHT maintains a minimalist overlay for safety only, but without long-range links. It rather relies on the physical neighborhood of the nodes traversed by a lookup request in order to spontaneously discover shortcuts in the logical space. This provides a form of liveness property, because long-range links may be encountered on a random basis. The conjecture is that lookup requests can be routed more efficiently and with much lower management costs than when deterministically maintaining $O(\log N)$ long-range neighbors.

Similarly to Chord and Pastry, the nodes are organized in a logical ring (Figure 2.1) where each node is assigned a random identifier in the logical space e.g., by hashing the node's IP using a cryptographic hash function such as SHA-1. Therefore, the physical neighbors of a given node are expected to be randomly distributed in the logical space as shown in Figure 2.1. This diversity property is important for the efficiency of the lookup algorithm. In addition, the node responsible for a key is the closest one in the logical space.

To limit the management overhead, each node n keeps track at all times of its successor $\text{succ}(n)$ and its predecessor $\text{pred}(n)$ in the ring. Additional robustness can obviously be obtained by considering several (logical) successors and predecessors.

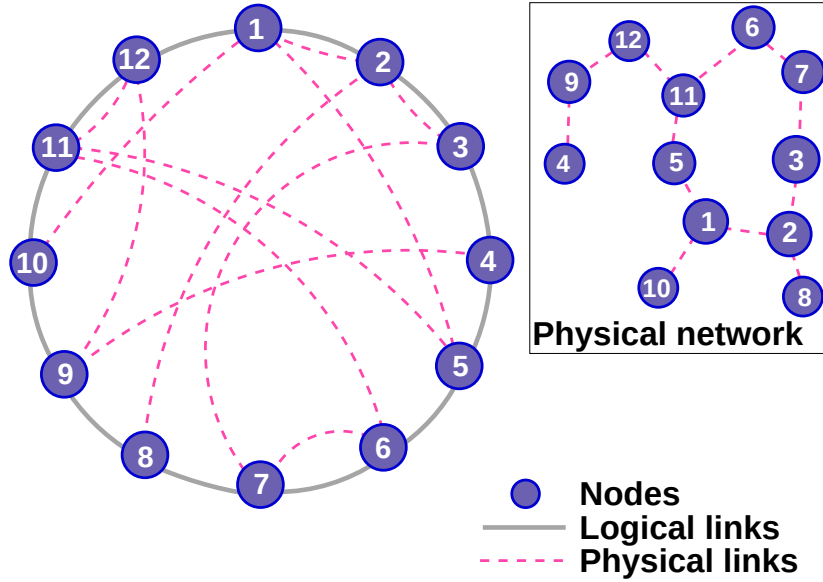


Figure 2.1: Illustration of the DHT model for ad-hoc networks

2.4 The basic algorithm

We present here the basic algorithm. For improved clearness, we will often refer to the pseudo code presented in Algorithm 1

To locate a key, a node creates a lookup message. This message contains the searched key (k) and the logical identity(id) of its current destination (n_d). Upon receiving a lookup message, a node n_i first searches among its physical and logical neighbors, itself and n_d the node with the smallest logical distance to the lookup key k (line 2). The logical distance is the distance on the logical ring between the lookup key and the current node's id. If n_i is closest, it is responsible for the key. Therefore, it has to reply to the originator of the request (line 4).

In this approach, the DHT lookup is closely integrated with the routing of messages through the ad-hoc network. Indeed, if there is a possibility to get a shorter logical distance to its final destination, the request might diverge from its original path at some intermediate nodes. Actually, if a node n_j , logically

Algorithm 1 Basic DHT lookup algorithm at node n_i for key k

 $\delta(k_1, k_2)$: distance between keys k_1 and k_2 $id(n)$: logical identifier (key) of node n V_i : physical neighbors of n_i $L_i \leftarrow \{pred(n_i), succ(n_i)\}$: logical neighbors of n_i

```
1: procedure LOOKUP( $k, n_d$ )  
    $k$  : looked up key  
    $n_d$  : next logical hop  
2:    $n_j \leftarrow \arg \min_{n \in V_i \cup L_i \cup \{n_i, n_d\}} \delta(id(n), k)$   
3:   if  $n_j = n_i$  then                                     { We are responsible for  $k$  }  
4:     return  $n_i$   
5:   else if  $n_j \in V_i$  then                                   { Go to physical neighbor }  
6:     send LOOKUP( $k, n_j$ ) to  $n_j$ ;  
7:   else if  $n_j \in L_i$  then                                   { Go to logical neighbor }  
8:      $n_k \leftarrow$  next step on physical path to  $n_j$ ;  
9:     send LOOKUP( $k, n_j$ ) to  $n_k$ ;  
10:  else                                                       { Continue to  $n_d$  }  
11:     $n_k \leftarrow$  next step on physical path to  $n_d$ ;  
12:    send LOOKUP( $k, n_d$ ) to  $n_k$ ;  
13:  end if  
14: end procedure
```

closest to the key is part of n_i 's physical or logical neighborhood, it will become the new destination. There are two cases to consider: if n_j is a physical neighbor of n_i the request is directly sent to that node (line 6); otherwise it will follow a multi-step path towards n_j (lines 8-9). On the other hand, if the remote node n_d is closer than any of the neighbors of a traversed node n_i , the request is simply forwarded to the next hop of the multi-step path to n_d (lines 11-12).

2.5 Convergence and termination

We study now the convergence and termination of the algorithm. For the remaining of this section, we consider that the system is in a stable state with no nodes joining or leaving.

To guarantee convergence, the algorithm always tries to reduce the logical distance from the next logical hop n_d to key k . Thus, as node n_i receives a request for the key k , it searches among its physical and logical neighbors, itself and n_d the node n_j closest to the key, thereby always trying to reduce the logical distance to k .

We then have four cases to consider:

1. If n_i is closest to k , then it is responsible for the key and the lookup ends.
2. If a physical neighbor is closest to the key, then it directly receives the request.
3. If a logical neighbor is closest to the key, then it becomes the new next logical hop n_d of the request.
4. Finally, if n_d is closest to the key, the current destination of the request is not changed.

Regarding these cases, the logical distance decreases in the first three ones, because either there is a node logically closer to the key than the actual best one, or the request reaches the node responsible for the searched key, hence ending the lookup.

In the last case, the distance to the key doesn't decrease, and the request is sent towards n_d . When the request reaches its current destination, one of the first three cases will apply.

This last case never applies forever, since the underlying ad-hoc routing protocol guarantees, that a request sent to a node will eventually reach that node. It follows, that the algorithm converges and terminates in a finite number of steps.

2.6 Increasing visibility

In the ad-hoc networks, nodes can only see their direct physical neighbors within their communication range, thus limiting the visibility of the physical neighborhood. A simple method to extend this visibility consists in exchanging (physical) neighborhood information. Thus, the nodes will not only know those located in their immediate communication range, but also the neighboring nodes of their neighbors. This extends the visibility to “neighbors of neighbors” (*NoN*).

The cost of this extension is limited in terms of message overhead, because the distribution of neighborhood information only requires a single broadcast message in case there is a change in the neighborhood. Clearly, as the number of long-range links increases, the probability to find a suitable logical shortcut during routing the request also increases. This has the potential to significantly improve the lookup efficiency.

2.7 Exploiting request history

MANETs exhibit two major characteristics. First, all the nodes have to participate in the forwarding and routing of messages. Thereof, a node “sees” traffic for which it is not the target. Second, as communication is broadcast, a node can listen to all the messages (called overheard messages in the following) sent by its physical neighbors.

By passively gathering the information transported around the network by the requests, one may now identify more potential long-range neighbors. To make this possible, a cache has been introduced in the algorithm. For each observed request (forwarded or overheard), a cache entry containing the key k and the destination of the message n_d is created. It uses a least-recently used replacement policy and has, in the evaluated implementation, its size limited to 256 entries. In the same way, the lifetime of the entries has been limited. They have the same lifetime as the concerned route maintained by the routing algorithm. At the end of this period (3 seconds in our implementation) and if not reactivated, the cache entry is simply discarded. A cache entry (k, n_d) can then be used by Algorithm 1 as long-range neighbor. When a node receives a request for key k_r , it also considers these cache entries when searching the node with the smallest distance to that key. If k is closer to k_r , the request is redirected towards n_d .

This extension is particularly interesting, because all the information about past requests can be passively gathered. Thus, absolutely no extra messages are required. Moreover, the memory used to keep the cache entries is precisely defined and limited.

2.8 Evaluation methodology

We present here the experimentation methodology that we used to evaluate the DHT algorithm. The same simulator and methodology are used to evaluate the multicast solution presented in Chapter 3. Thus, only a recall will be done in the evaluation part of that chapter.

We also implemented the DHT algorithm in the Freemote emulator [fre 2009].¹ The emulation methodology, the system configuration and the results will be presented in parallel with the simulation evaluation.

2.8.1 Experimental simulation environment

We consider that the DHT algorithm runs on resource limited devices used by people in public spaces or in office buildings for professional and entertainment purposes. We may thus expect that all participating devices will have on average 13 to 16 direct neighbors. By neighbor, we designate nodes that are located in the communication range of each other and being able to communicate together.

Only bidirectional links are considered, thus we consider that the unidirectional communication property is dealt with and hidden at the network layer. Communication is broadcast, thus all the nodes in the communication range of a transmitting node can listen to all transmissions. Every node in the network runs the DHT algorithm and has the ability to gather information from relayed and overheard messages as well as to influence the routing of relayed messages.

We experimented only with simple scenarios. Nodes are placed randomly (according to a uniform distribution) in a rectangular area. The edge of the Cartesian area is computed dynamically with the regard of the desired density.

¹These experiments have been done by Timothée Maret.

In our simulation configurations, the edge is 1414 meters for the 1000 nodes network and 3162 meters for the 5000 nodes network. The presented results are issued from static and mobile configurations. In the static scenarios, the nodes are placed at the beginning of the simulation, and their positions do not change anymore. In contrast, for mobile configurations, each node can decide to stay or to move for a randomly chosen period of time t . We thus consider that all participating devices (nodes) are uniformly distributed in a Cartesian space where they can independently decide to move during a finite period of time t with a speed S randomly chosen in the interval $0 \leq s \leq 2m/s$ in arbitrary directions to reflect human displacements. If a node reaches the border of the simulation space, its direction is altered so as to continue its displacement inside the space. All the nodes involved in the simulations always form a connected graph.

To support the DHT, we simulated the ad-hoc on-demand distance vector (AODV). This routing protocol is able to build a multi-step path between any of the nodes of the network and to route messages between them. The DHT communicates with the routing protocol through dedicated methods. AODV provides also a method to the DHT to intercept all the requests traveling through the node.

The DHT identifiers are randomly assigned to nodes. Warm-up sequences are used with the algorithm using a cache in order to populate the routing tables before evaluating the DHT. The lookup performances have been obtained with 2000 lookups issued from random nodes searching for randomly chosen keys.

We experimented with various configurations and network sizes:

- *Network sizes* : 1,000 (1K); 5,000 (5K);
- *Communication range*: 100 meters;
- *Simulation space edge*: 1414 meters(1K nodes), 3162 meters(5K nodes);
- *Connectivity* : the average number of physical connections of a node (network density) varies between 13 and 16;
- *Lookup requests* : for each experiment, the paths of at least 2,000 randomly generated requests are statistically evaluated;
- *Steadystate* : to evaluate the variant of the ad-hoc lookup algorithm using the caching mechanism, the simulation runs a warm-up phase during 2000 (2K) requests in order to reach a steady state before the statistical information is collected for the analysis;
- *Mobility* : randomly chosen in the interval $0 \leq s \leq 2m/s$ in random directions during a finite time t .

The following versions of the DHT lookup algorithm have been evaluated:

- *Basic*: The basic DHT lookup;
- *Neighbors – of – neighbors (NoN)* : the basic algorithm evaluating the physical neighbors and their neighbors to choose the next step;
- *Cache (C)* : the NoN algorithm using a cache to memorize previous forwarding choices;
- *Warm – up (Wup)* : to allow comparisons, the simulation system executes a warm-up phase to reach a steady state prior to issuing the analyzed requests.

We evaluated the percentage of altered paths (i.e., shortcuts taken) to determine how often the shortcuts are used. We also evaluated the average number of physical steps, the number of logical hops to achieve a lookup, as well as the cost of a logical hop to obtain a performance evaluation in terms of physical complexity and to compare with other approaches. Then, we evaluated the percentage of achieved logical steps relatively to the distance to the node responsible for the searched key in order to determine when shortcuts are used in the requests' paths. Finally, we evaluated the request stretch. This measurement shows the difference between the average physical path length of the lookup and the number of physical steps on the direct route between the requester and the requested node.

2.8.2 Experimental simulation results

Logical shortcuts

The logical paths of a lookup are evaluated by the algorithm at each physical step during its routing toward the desired key k . An intermediate node always forwards the request to the node featuring the smallest logical distance to the destination of the request. Thus, a logical path may be altered thanks to the shortcuts in the logical space provided by the physical neighbors, the neighbors of the neighbors, and the cached routes. Figure 2.2 presents the average number of logical paths started relatively to the number of logical paths terminated for different network sizes (e.g., 5K nodes and 1K nodes) and different network configurations (e.g., static and mobile configurations).

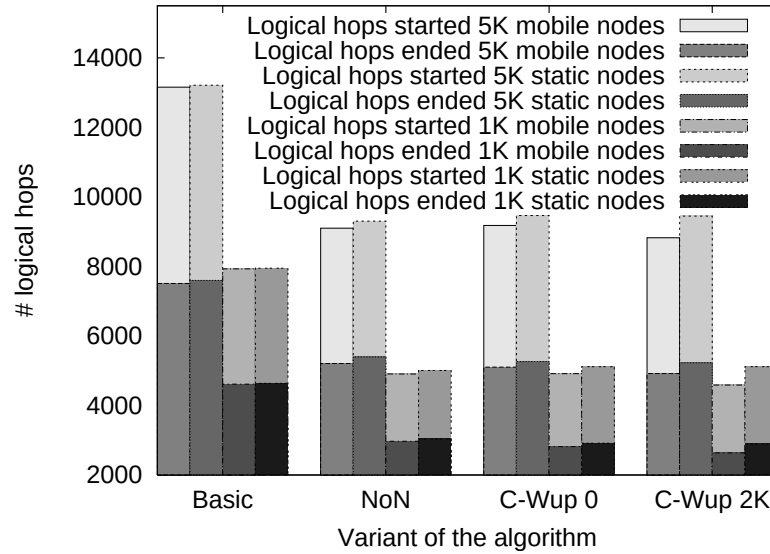


Figure 2.2: Comparison of the number of logical hops started and terminated

We can see that in all the tested configurations, more than 40% of the logical paths started didn't succeed, because an intermediate node has found a

shortcut in the logical space. This is confirmed in Table 2.1 that presents the details of the use of shortcuts during logical hops. Figure 2.2 describes another interesting observation. Indeed, the basic algorithm requires around 30% more logical hops than all the other versions. As a request is sent through a logical hop only if no other possibilities exist, this means that the lookup algorithm uses directly logical shortcuts to route a request and avoids relying on expensive logical routes when possible. This demonstrates, as expected, that the improvements are effective in providing more logical shortcuts.

Network size & type	Basic	NoN	C-Wup 0	C-Wup 2K
1,000 - static	41.84%	39.46%	43.07%	43.5%
1,000 - mobile	41.83%	39.76%	42.73%	42.5%
5,000 - static	42.49%	42.57%	44.39%	44.66%
5,000 - mobile	42.93%	42.78%	44.4%	44.26%

Table 2.1: Average use of logical shortcuts

As the algorithm relies on the close neighborhoods and the cached requests to minimize the logical distance to the destination, intuitively, this suggests that more and better shortcuts can be found at the beginning of a logical path. Indeed, the probability to find a logical shortcut decreases when approaching to the destination. Therefore, the probability that logical hops terminate without a shortcut being taken increases when approaching the request's destination. This characteristic is demonstrated in Figure 2.3. The presented results confirm that the percentage of terminated logical hops increases when getting closer to the node responsible for the key.

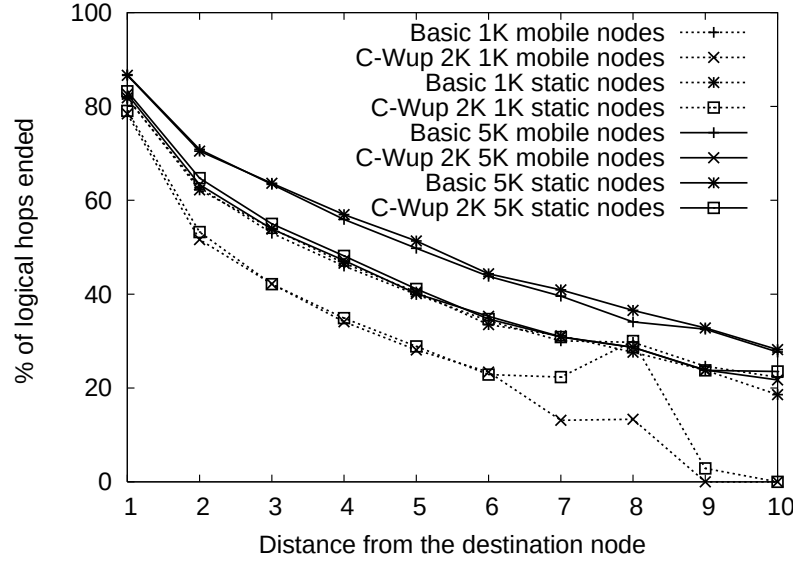


Figure 2.3: Percentage of terminated logical hops

Costs of a request

Figure 2.4 presents the average number of physical steps required to serve a request. The DHT lookup algorithm supports well the mobility, because the average number of physical steps is almost the same for both configurations. Moreover, the improved algorithm requires roughly 30% less physical steps to achieve a request than the basic version, thus confirming their efficiency. This is shown in Figure 2.5 presenting the average number of logical hops required to reach the node responsible for the searched key. The improved algorithm uses 30% to 40% of logical hops less than the basic algorithm. Thus, by reducing the number of logical hops, the lookup algorithm reduces the total number of physical paths required to achieve a lookup. Table 2.2 confirms this result. Because the cost of a logical hop in terms of physical steps is almost the same for all tested configurations, it is necessary to reduce their usage to reduce the total number of physical steps.

Network size & type	Basic	NoN	C-Wup 0	C-Wup 2K
1,000 - static	10.72	11.17	10.93	11.06
1,000 - mobile	10.88	11.47	11.25	11.75
5,000 - static	23.01	23.49	23.01	23.14
5,000 - mobile	23.44	24.02	23.36	23.7

Table 2.2: Average cost of a logical hop in terms of physical steps

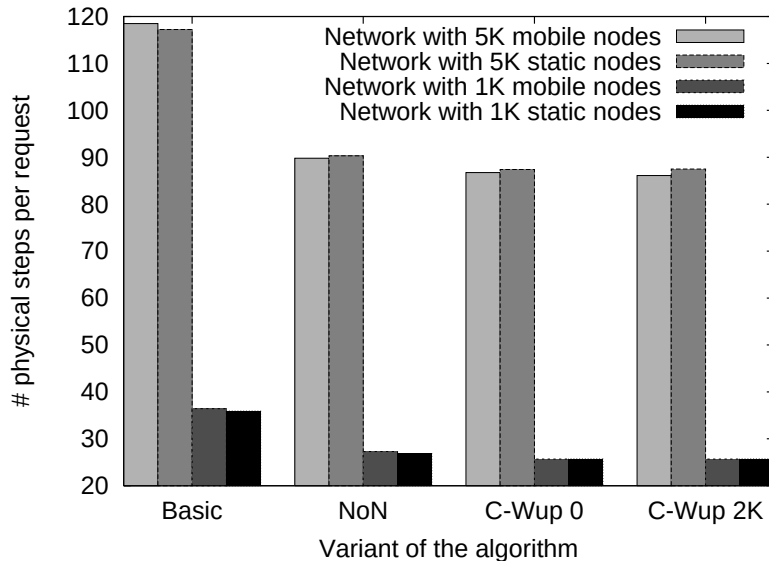


Figure 2.4: Average number of physical steps to achieve a request

All results listed demonstrate that the NoN improvement is the most effective method, and that the caching effect is limited. Clearly, the efficiency of the cache improvement is directly linked with the renewing policy. In the presented experiments, the cache entries are discarded as soon as the route to the memorized destination expires. Thus, most of the cache entries had a life-time limited to 3 seconds (the value chosen in the implementation) as the routing algorithm discards older routes. The cache efficiency can be easily improved by keeping the entries longer.

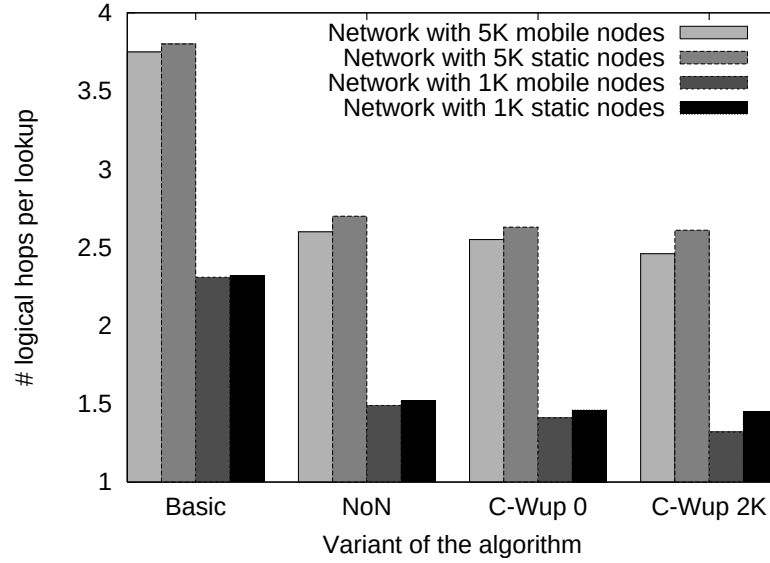


Figure 2.5: Average number of logical hops per lookup

In Figure 2.6, we observe that our algorithm always achieves the lookup in less than $\log N$ hops or even less than $\frac{1}{2}\log N$ hops. This clearly demonstrates the efficiency of the approach and confirms that the introduction of the improvements further ameliorates performance. Moreover, it must be noted that the average search cost using the logical links has only a performance of the order of $O(N)$. In comparison, Figure 2.6 clearly shows the superiority of exploiting the properties of the physical links in the algorithm and the scalability of such approach.

Finally, Figure 2.7 exposes the path of a request in an ad-hoc network using the algorithm with all improvements.

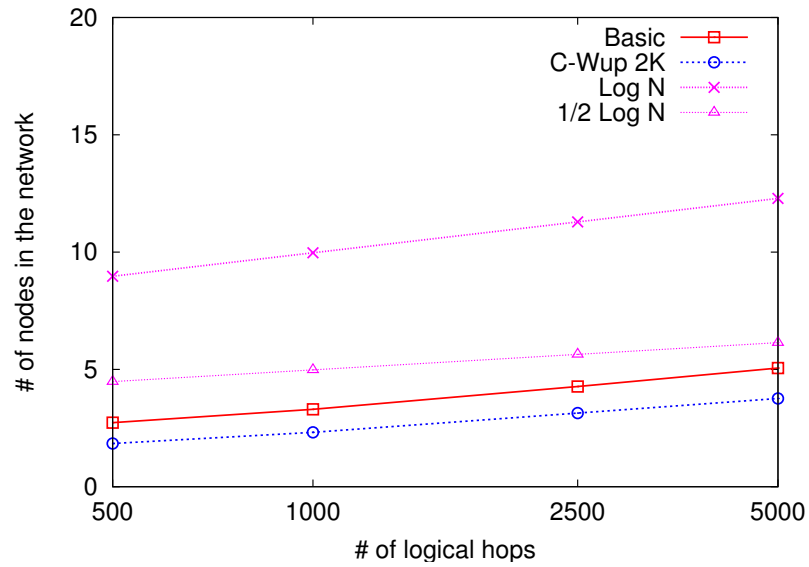


Figure 2.6: Average number of logical hops per lookup in different network sizes

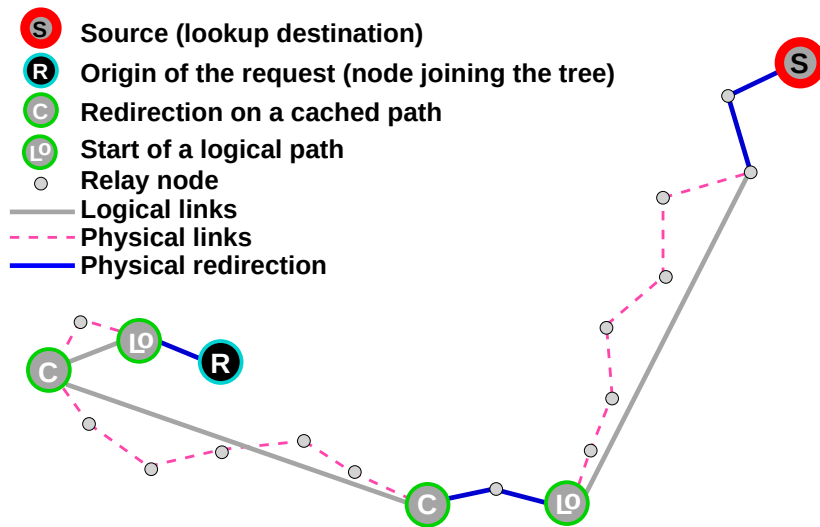


Figure 2.7: Path of a request routed by the DHT with all extensions

Request stretch

In Figure 2.8 we present the stretch of the DHT request. We measured the difference between the average number of physical steps required to achieve a request and the average physical steps on the direct path between the requester and the node responsible for the key.

As for all other measurements, the improved versions of the DHT lookup algorithm perform significantly better than the Basic one. The NoN improvement produces the biggest stretch reduction. This is not surprising as in all previous measurements, this extension provides the greatest performance improvements. The improvements allow to reduce the global stretch from around 25% on all networks and configurations.

Figure 2.8 shows as well that the stretch increases with the network size. This result is not surprising, as the network with 5000 nodes has the same density as the one with 1000 nodes. Thus the potential number of shortcuts provided by the NoN and Cache improvements is approximatively the same, however, the network is 5 times bigger.

Finally, the number of physical steps to achieve a request is around 2.5 times the direct way between the requester and the requested node for 1000 nodes network and 3.5 the direct way for the 5000 nodes network. This confirms that the path followed by a request during the lookup procedure is quite short and converges quickly to the node responsible for the request. This is also confirmed in Figure 2.7 presenting a request's path routed by the DHT with all extensions.

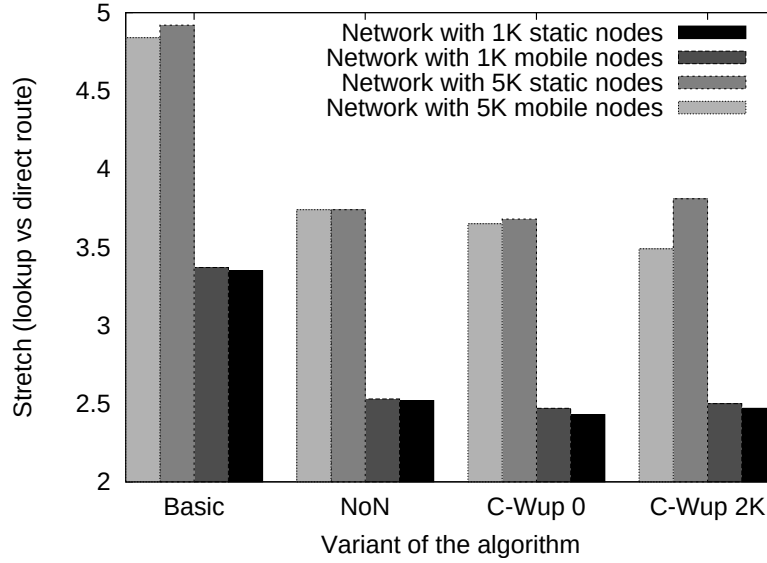


Figure 2.8: Stretch of a DHT request vs direct route

2.8.3 Experimental emulation environment

The Freemote emulator has been used in order to validate the performance of our ad hoc DHT (Distributed Hash Table) algorithm. All the details on the freemote architecture and its operation mode can be find in [Appendix A](#)

In this section, the simulation results presented in [\[Kummer 2006a\]](#) serve as a reference when evaluating results obtained in the Freemote emulator mixing real and emulated Java nodes. Indeed, the setup used in the Freemote emulation is based on the setup used in these simulations and is bit different than the one used to produce the measurements presented in [Section 2.8.2](#). The two main metrics used in comparing results are:

- average percentage of logical shortcuts to determine their usage frequency
- average number of physical steps per request to compare performance in terms of physical complexity

These metrics are measured in 3 different configurations:

1. *Basic*: basic DHT lookup
2. *Neighbors-of-Neighbors (NoN)*: basic algorithm that evaluates physical neighbors and their neighbors in order to choose the next step.
3. *Cache (C)*: *NoN* algorithm uses a cache to memorize a finite history of previous forwarding choices. The Cache (*C*) configuration includes the neighbors-of-neighbors (*NoN*) extension.

The simulation results measured in Section 2.8.2 demonstrate that the Cache version always performs best. Although not as efficient as the DHT with Cache, in all cases the NoN version outperforms the Basic algorithm.

For these tests the Freemote emulator environment was deployed on 6 JMotes² and up to 10,000 emulated motes. Ten 2.7 GHz CPU/2GB RAM standard machines running Windows XP SP2 were used over a 100 Mb/s switched Ethernet LAN. One computer was dedicated to the (emulated) aggregation node, the bridge node and the Topology Manager server. All other emulated nodes were equally distributed across the nine other machines. Each computer was also running a Topology Manager client. The emulated nodes were randomly distributed in the simulation space: a simple square. The average connectivity was 14 and the DHT based overlay network was implicitly constructed: each node computed its key in the logical space based on its routing address. Hence, the logical space a node is responsible for, as well as its logical neighborhood, were provided by the node's position on the ring.

²JMotes are CLDC Java based mote prototypes developed at the University of Applied Sciences in Fribourg [eia]

The following parameters differ from the reference simulations [Kummer 2006a] and thus explain certain differences when comparing results.

- Routing algorithm: in the Freemote emulator the NST-AODV [Gomez 2006] algorithm is used instead of the simplified LAR (Location Aided Routing) [Ko 2000] applied in the reference simulations. The simplified LAR algorithm used takes advantage of the knowledge assumed about the node's geographical position in order to send messages to the physical neighbor closest to the destination node, and LAR is able to obtain shorter routes. Our NST-AODV implementation was tested with real and emulated nodes in different scenarios including mobile configurations. Experiments detailed in [Borgia 2007] showed that the classical AODV [Perkins 1999] algorithm performed poorly over asymmetric links, because it assumed symmetric links when building the routing paths. This issue is discussed in the AODV literature and so we decided to implement the black list solution in our NST-AODV for handling of asymmetric links.
- DHT key management: The implemented DHT algorithms use a non-standard scheme to manage keys: nodes are placed at the beginning of their key spaces rather than in the middle. A node is thus responsible for a key that is not necessarily the closest one. This scheme is used to solve the problem of keys being exactly at the same logical distance from two consecutive nodes (i.e., the same logical distance with the previous and the next node on the logical ring). Thus, a node is always responsible for all the IDs included in the logical subspace, starting at its own node ID and up to the node ID preceding its direct successor.

The following section describes and compares the results of the two above-mentioned metrics, obtained using the dedicated simulator and the Freemote emulator.

2.8.4 Experimental emulation results

Average percentage of logical shortcuts

Figure 2.9 shows the percentage of found logical shortcuts during the request routing. It can be seen that the DHT algorithm's Freemote emulation shows a behavior similar to that of the simulated one. In both systems, more logical shortcuts were found when extensions were applied, the number of logical shortcuts found increased proportionally to the network size, except for the emulated cache version. This was expected because for a greater number of nodes, longer physical routes are required, and thus more shortcuts are found. The smaller percentage of logical shortcuts found by the DHT simulated cache version of the algorithm was due to the underlying routing methodology, which may route a request to a node where no further neighbors can be found in the following step. Thus, to avoid a costly backtracking process, the request was directly sent to the destination node, costing an average logical step, and consequently for this process no logical shortcut could be found. This simulation choice also explains the smaller percentage of logical shortcuts found by simulated algorithm in the 10,000 nodes network.

Nb of Nodes	Basic			NoN			Cache		
	Freemote	ref.	Δ	Freemote	ref.	Δ	Freemote	ref.	Δ
100	30%	<i>N/A</i>	<i>N/A</i>	19%	<i>N/A</i>	<i>N/A</i>	26%	<i>N/A</i>	<i>N/A</i>
1,000	41%	38%	3%	41%	37%	4%	52%	51%	1%
10,000	50%	39%	11%	51%	41%	10%	64%	49%	15%

Table 2.3: Average percentage of logical shortcuts *N/A* = Not Available

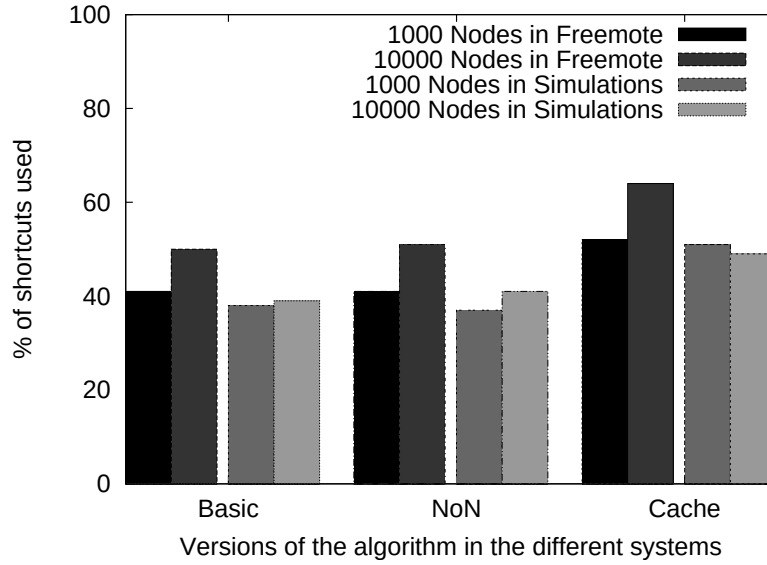


Figure 2.9: Average use of logical shortcuts during a request’s logical path

Since the results for the Freemote emulators correspond to those found in the reference simulations, they tend to validate the emulations with numerical differences (Δ) shown in Table 2.3, ranging from a few percent for 1,000 nodes to a maximum of 15% for the 10,000 nodes networks.

Average number of physical steps per request

As illustrated previously by comparing the percentage of logical shortcuts found in the emulated and simulated DHTs, Figure 2.10 shows that all algorithms’ implementations result in the same behavior for all network sizes. When applying extensions, the number of physical steps needed to achieve a lookup decreases. This is not surprising, given that the purpose of these extensions is to provide more logical shortcut opportunities. Thus, by providing more shortcuts, the algorithm reduces the number of logical hops per lookup and reduces the average number of physical steps.

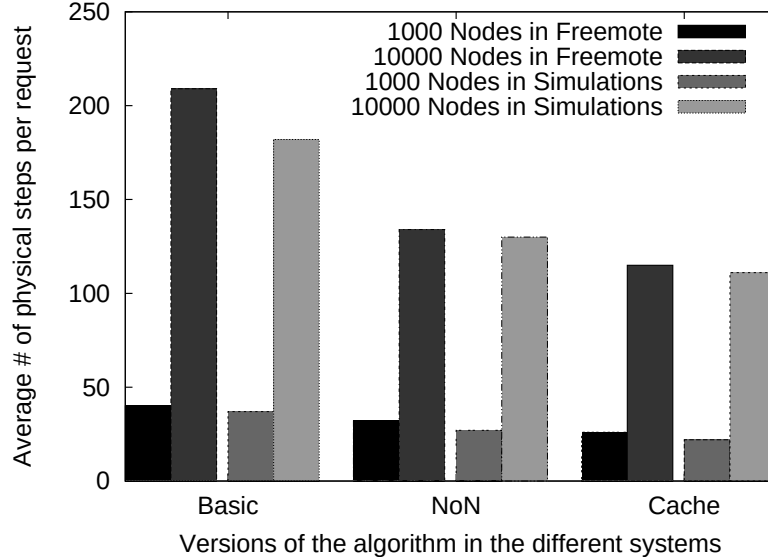


Figure 2.10: Average number of physical steps to achieve a request

As shown, the number of physical steps is greater, which can be explained in two ways: 1) Different key maintenance schemes are used. The nodes are not placed in the middle but at the beginning of the key space, and on average at least one more logical link is required to reach the node responsible for a DHT key. 2) Different routing schemes. As detailed above, when blocked at an intermediate node a request may be directly delivered to its next destination. When this rescue routing process is applied, the average physical cost of a logical hop is counted, this way perhaps slightly reducing the overall average physical cost of requests.

Table 2.4 again shows that the results obtained using the Freemote emulator are in line with the simulations. In fact, the numerical differences (Δ) are all below 21%, meaning satisfactory results given the differences between the implementations and the underlying routing methodology.

Nb of Nodes	Basic			NoN			Cache		
	Freemote	ref.	Δ	Freemote	ref.	Δ	Freemote	ref.	Δ
100	7.33	<i>N/A</i>	<i>N/A</i>	6.89	<i>N/A</i>	<i>N/A</i>	6.50	<i>N/A</i>	<i>N/A</i>
1,000	40	37	8.2%	32	27	19.1%	26	22	21%
10,000	209	182	14.9%	134	130	3.1%	115	111	4.1%

Table 2.4: Average number of physical steps per request

Nb of Nodes	NoN			Cache		
	Freemote	ref.	Δ	Freemote	ref.	Δ
100	6%	<i>N/A</i>	<i>N/A</i>	11.3%	<i>N/A</i>	<i>N/A</i>
1,000	19.7%	27%	-7.3%	33.6%	40.5%	-6.9%
10,000	36.1%	28.8%	7.3%	45%	39.3%	5.7%

Table 2.5: Average gain in physical steps compared to “Basic” DHT, for NoN and Cache DHT

Finally, Table 2.5 shows the average reduction in the number of physical steps for the NoN and Cache DHT implementations compared to the Basic DHT version. As expected, the DHT algorithm extensions make it possible to reduce the request routing costs. Again the results of both implementations are close, with the Freemote emulator and the reference results lying within a 10% margin.

In a general, all the above comparisons show that the Freemote emulator’s results fall within a few percentage points of those for simulation, usually less than 15%. A worst-case discrepancy of 21% can be explained by the differences between the routing algorithms. The variations in results are due to the differences between the two setups.

2.9 Summary and discussion

In this chapter, we presented our DHT algorithm. It combines a minimalist logical overlay structure together with adaptive routing mechanisms to quickly locate content. Nodes are organized in a logical ring, similarly to Chord [Stoica 2001] or Pastry [Rowstron 2001a]. The long-range links are found in the direct physical neighborhood of the nodes traversed during the ad hoc routing. Thus, their cost in term of maintenance and routing (i.e., physical steps) are limited. In addition, we developed some extensions that provide an extra level of visibility in the physical neighborhood. They provide additional long-range links to the DHT, thus participating in the lookup efficiency. This design improves lookup performance and reduces the message overhead as compared to the flooding-based protocols which yield linear or worse performance.

Our simulation results have demonstrated that a sub-linear lookup performance can be preserved when relying on MANETs particularities to discover logical shortcuts. Despite certain differences, it was shown that similar trends and numerical values could be obtained in the emulation. Typical differences between these two results were only within a few percentage points, explained by different routing algorithm and methods: NST-AODV for the Freemote emulator and a simplified LAR (Location Aided Routing) for simulations. Finally, the same trends (i.e., average percentage of logical shortcuts, average number of physical steps per request, average gain in physical steps) are observed between all results, independently of implementation and the routing algorithm (LAR, AODV, NST-AODV) used to support the DHT algorithm.

It must however be noted that, to the best of our knowledge, the observed sub-linear performance still has to be confirmed by an analytical study. It will also be helpful to determine the limits of such approaches.

This DHT is able to support many applications. Thus, an algorithm for building multicast trees in mobile ad-hoc networks using this specialized

lightweight DHT overlay has been developed. The DHT is used to efficiently localize the root node of the multicast tree. This algorithm is described in the following chapter.

Building multicast trees in MANETs

3.1 Introduction

As for the information localization problem discussed in the previous chapter, solutions based on flooding cannot attain acceptable performance levels when distributing content to a subset of interested nodes. Indeed, in the MANET approach, many nodes are forced to participate, even when not interested in the distributed content. This leads to ineffective solutions and poor performance, with the network quickly becoming overloaded and consuming a lot of energy and bandwidth.

Thus, in this chapter, we propose a novel algorithm for building multicast trees in ad-hoc networks. We use several techniques to connect joining nodes to an existing group member that is physically close and on the request's path to the source. We propose various extensions to reduce the number of relay nodes but not members of the multicast group implicated in the relaying of messages. This algorithm is superimposed to the MANET DHT presented in Chapter 2. As in Scribe [Rowstron 2001b] and XScribe [Passarella 2006], the DHT is used for efficiently localizing the source of the multicast tree and one tree is created per source. The proposed algorithm is particularly well adapted to wireless sensors networks. In this environment the mobility is limited thus causing only few membership changes.

We evaluated our algorithm by the means of extensive simulations. Different scenarios were used but again without taking into account churn (nodes frequently joining and leaving the ad hoc network). Results indicate that our algorithm does produce efficient multicast trees, with a limited number of non member nodes relaying the multicasts, and that it scales well to large networks.

The remainder of this chapter is organized as follows. Section 3.3 presents the basic idea supporting our algorithm. Section 3.4 introduces the terminology that will be used in this chapter and presents the general concept of our algorithm. We then detail the basic algorithm in Section 3.5. Sections 3.6, 3.7, 3.8 and 3.9 present the extensions of our basic algorithm. We detail the evaluation methodology in Section 3.10 and present the experimental results in Section 3.11. Finally, Section 3.12 concludes.

3.2 Related work

Several P2P approaches have been suggested for multicasting in ad-hoc networks. Some use a logical overlay substrate for source localization while others rely on flooding. We only discuss a selection of the various approaches known from literature to present only closely related approaches (see [Chen 2003] for a good survey).

Different logical structures can facilitate appropriate tree construction in ad-hoc networks. MZR [Devarapalli 2001] relies upon the Zone Routing Protocol [Haas 1997] to build a multicast tree. The nodes in ZRP define a zone around them and proactively maintain routes to all nodes within that zone. A reactive route discovery protocol is used when the destination is outside the sender's zone. When a source has data to multicast, it advertises it to all the nodes in its zone, and then extends the tree to nodes at the border of other zones. An interested node has to answer to the source and, when the message reaches a multicast group member, a branch is created. While the zone structure

contains the flooding necessary to build the tree, it still floods the whole network zone by zone. Its bandwidth and energy requirements are thus significant. In addition, this protocol doesn't provide generic lookup facilities as it is done in our algorithm by the DHT.

XScribe [Passarella 2006] and Georendezvous [Carvalho 2006] use a DHT to support the multicast tree creation. XScribe is based on Cross-ROAD [Delmastro 2005], a cross layer DHT providing the same features as Pastry, but based on a proactive routing protocol and with lower bandwidth requirements. XScribe exploits the DHT routing capacities to distribute multicast messages. Each source has to know all the members of the group and sends the multicast messages directly to each member by unicasting messages. Therefore, the approach does not scale well, nor does it try to optimize the resource consumption (by minimizing the number of relay nodes or the number of transmissions required).

Georendezvous relies on CHR [Araujo 2005], a specialized ad-hoc DHT that groups nodes in clusters according to their physical location. The DHT is used to efficiently locate the cell responsible for a group. The nodes in this cell manage the membership for the group and forward the multicast to all the members. Membership management is centralized in a cell containing multiple nodes, which are also responsible for distributing multicast messages. This approach may suffer from high bandwidth and energy requirements thus offering poor scalability.

Jetcheva and Johnson proposed the Adaptive Demand-Driven Multicast Routing Protocol (ADMR) [Jetcheva 2001]. This approach aims to reduce as much as possible the not on-demand components of the protocol. Source based tree are created as soon as there is at least one source and one member in the network. The algorithm uses monitored traffic around the source to determine link breaks or if a source has to stop sending multicast packets. The

multicast packets are forwarded between tree members. The algorithm also integrates additional mechanisms to locally repair the multicast tree in case of disconnection and to clean the tree structure.

Chen et al. implemented the ADMR algorithm on CC2420-based nodes using TinyOS operating system [Chen 2006]. They evaluated the performance of the algorithm in this environment and proposed some adaptations. They proposed a new metric that estimate path delivery ratio and optimize it rather than the minimizing the number of hops. They also improve the route reinforcement, and proposed a passive and an active route reinforcement method. The active route reinforcement method requires that a node continue to receive message in order to stay alive. On the contrary, the passive route reinforcement method makes nodes staying active as long as they receive or overhear messages.

In [Ge 2006], the ALMA (Application Layer Multicast Algorithm) is proposed in order to evaluate the efficiency of application layer multicast algorithm in mobile ad hoc networks. ALMA creates root based multicast trees. The nodes try to connect to physically close members. The subset of candidates nodes is provided by either a rendez-vous node or local search. Additionally methods to rearrange the tree in case of bad link quality between nodes are provided. Here the quality of the link is evaluated in terms of Round Trip Time (RTT)

More recently, Koutsonikolas et al. proposed the Hierarchical geographic multicast routing for wireless sensor networks (HGMR) [Koutsonikolas 2010]. This algorithm combines two previous approaches, the Geographic multicast routing for wireless sensor networks algorithm (GMR) [Sanchez 2006] and HRP (Distributed hashing for scalable multicast in wireless ad hoc networks)[Das 2008]. The idea behind HGMR is to take advantage of the key design of both algorithms to optimize both, forwarding efficiency and as well as scalability to large networks. HGMR uses the mobile geographic hashing concept from HRP to decompose the multicast group into smaller subgroups

of manageable size. Then, in each subgroup, the local multicast scheme of GMR is used to forward messages among the multicast tree. The HRPm space division approach brings scalability in HGMR while the GMR transmission approach improves the forwarding efficiency.

The Geographic-aided Multicast Zone Routing Protocol (GMZRP) [Cheng 2009] is an on-demand multicast protocol which builds the multicast tree when required. It is assumed that the devices running the algorithm are equipped with GPS devices. The algorithm uses this geographical information to estimate the center of the network where the partitioning is initiated. When a source node has data packets for a multicast group, it initiates a tree establishment process. During the process, the geographic partition of the network is used to ensure that each zone will be queried only once, thus limiting the tree discovery overhead.

Finally, the Topology-Aware Group Communication Middleware for MANETs [Garcia Lopez 2009] operates OMOLSR, an overlay multicast protocol on top of JOLSR, a Java implementation of the MANET OLSR [Clausen 2003] routing protocol. OMOLSR uses the network graph generated by JOLSR to compute a virtual mesh connecting all members of a multicast group. Then when the graph is ready, the minimum spanning tree can be computed by the source. The tree build can then be used to route the multicast messages to all the members of the multicast group. The main characteristic of OMOLSR is that it uses the topology information gathered by JOLSR thus not needing to send additional packets.

3.3 Design rationale

The first interesting question that one may ask is why do we investigate an approximation method of the best tree and not directly try to find an optimal solution? In principle, the networks we consider in the chapter can be formalized as undirected graphs where the vertices (nodes) are placed on a two dimensional plane. The nodes have the same transmission range r and there is an edge between two vertices of the graph if their physical distance is smaller than r . In this model, we consider only bidirectional links, however unidirectional links may be used, we consider those being hidden by the network layer.

The simple model considered for our experiments is thus quite general. It perfectly matches the one considered in [Ruiz 2005] by Ruiz et al. They demonstrated that finding minimal cost trees given a source s and a set of receivers R in this kind of graph is NP-hard. Thus, solving this problem requires approximation method like the one presented in this Chapter.

As described in Section 3.10.1, 10% of the nodes are tree members. Indeed, if a large amount of the nodes in the ad hoc network were members, a broadcasting approach would be recommended. A simplistic approach to deliver multicast content to all subscribers consists of separately unicasting each member. In this way, each member is a leaf member and receives multicasts directly from the source through the physically shortest path. This kind of approach requires many transmissions (i.e., resources in general) and relay nodes to distribute the multicast messages. Consequently, we investigate an algorithm joining the messages' distribution paths to limit the number of relay nodes required to forward the desired content.

More precisely, there is a possibility to build a tree including less relay nodes and requiring less transmissions than the unicast shortest path tree described above as soon as there is more than 4π members.

First, we consider n multicast members uniformly physically distributed around a

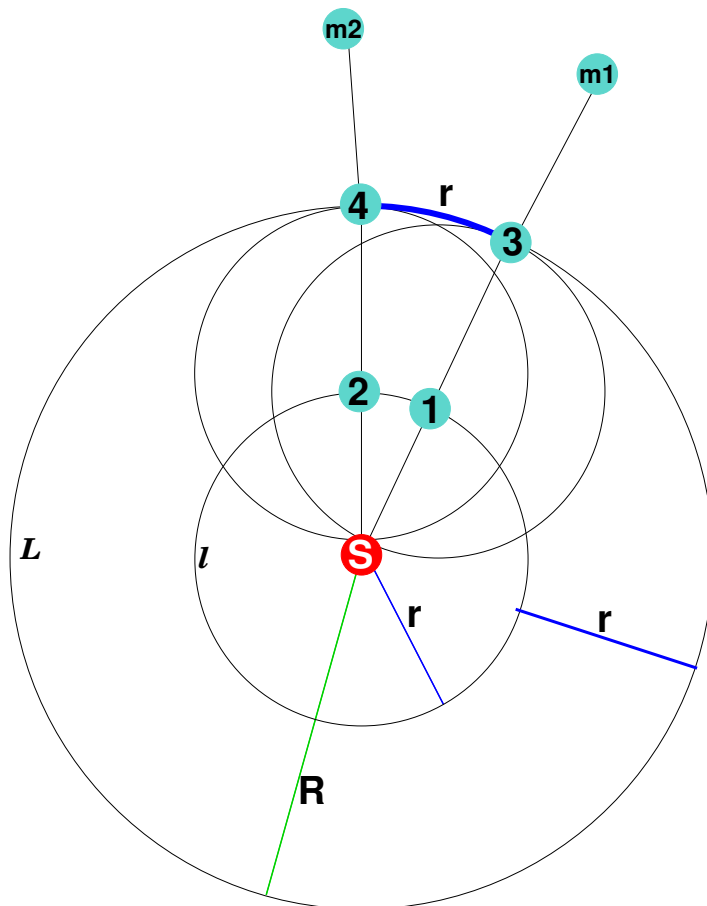


Figure 3.1: Illustration of the utility of our algorithm.

source s . Let's take as an example the limited situation presented in Figure 3.1 for better understanding. As the group members are uniformly distributed around the source, the presented case can be reproduced all around the source. The communication range of the nodes is r , i.e., two nodes n_1 and n_2 can communicate if $|n_1 - n_2| \leq r$. In Figure 3.1, the source s can directly communicate with nodes 1 and 2, 1 and 2 with $\{2, 3\}$ and $\{1, 4\}$ respectively. Finally, there is a communication link between 3 and 4. We consider a member m_1 reachable through nodes 1 and 3 and a second one, m_2 through 2 and 4. There exists a route to reach m_1 and m_2 passing by relay nodes 1, 3 and 4, thus removing one

relay node and reducing the number of transmissions as soon as there will be enough members uniformly distributed around s such that the distance between the relays located on the second circle of length L will be $\leq r$. Let's define the number of members uniformly distributed needed so that the relay nodes on their path from s on circle L will be separated by a distance $\leq r$ as N . We have that $R = 2r$. We know that $|L| = 2\pi R \Rightarrow |L| = 2\pi(2r) \Rightarrow 4\pi = \frac{|L|}{r}$. Thus, L can be covered with $4\pi * r$. Consequently, if $N \geq 4\pi$, routes can be merged, hence less transmissions and relay nodes are required to reach all the members. Second, if members are not uniformly distributed, it is possible that the distance between two relay nodes on L is bigger than r . As the relay nodes are on circle L , there is at least one location around the circle L where the distance separating two relays is $\leq r$, thus allowing to merge paths in order to reduce the transmissions and the number of relay nodes. $\square$

3.4 General concepts

Before describing our multicast algorithms, we introduce the terminology that will be used in this section. Table 3.1 indicates the terms used for the nodes running the algorithm relatively to their role. The different terms are also illustrated by means of a sample multicast tree in Figure 3.2.

Term	Definition
Nodes	All the nodes in the system.
Members	All the nodes interested in a multicast group.
Non-members	All the nodes not members of a multicast group.
Internal members	Members inside the tree helping to distribute multicast messages (i.e., members with children).
Leaf members	Members at the end of a branch of the multicast tree (i.e., members without children).
Relay nodes	Non-member nodes relaying multicast messages.
Non-relay nodes	Non-member nodes not included in multicast tree activities.

Table 3.1: Terminology

In MANETs, the distribution of information to a group of nodes can be achieved essentially in two ways: flooding the network to locate the source and building a tree as done for example in MAODV [Royer 1999] or locating the source and building the multicast tree using directed search algorithms as done for instance by Scribe [Rowstron 2001b]. Unfortunately, flooding methods involve all the nodes in the network whether they are interested or not in the distributed content. Flooding methods also suffer from poor scalability as a lot of messages are sent through the whole network. In contrast, directed search based multicast methods are less energy and bandwidth demanding.

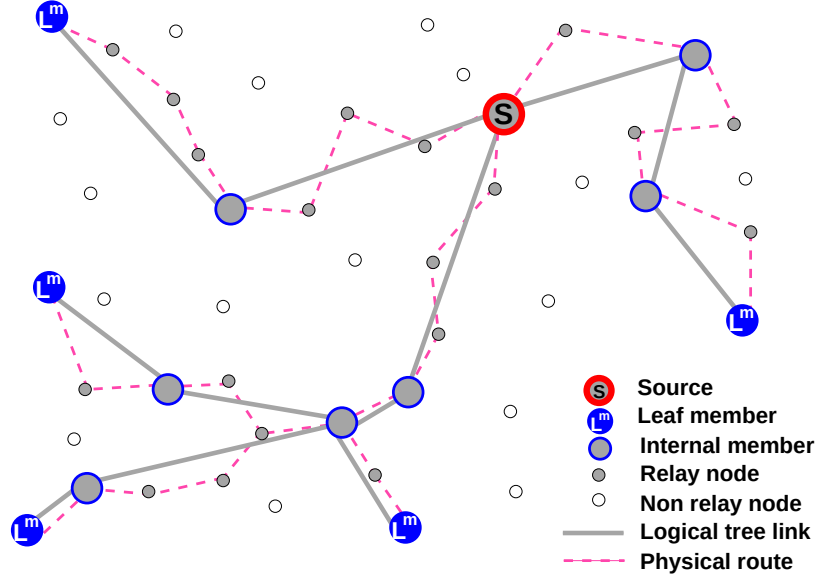


Figure 3.2: Sample tree illustrating the used terminology.

Alas, these methods cannot be simply reused in the MANETs context, because they do not consider the main specificities of the underlying network like multi-hop communication and physical awareness.

The algorithm presented here builds a multicast tree for each group. To join a multicast group or to send a message to all its members, a node has to first lookup the data source acting as “rendez-vous” point. The multicast algorithm is supported by the DHT presented in Chapter 2 and provides the necessary efficient lookup facilities for MANETs. Moreover, the multicast algorithm may leverage from the scalability properties of the DHT.

Although the lookup messages may not follow the shortest path in the underlying ad-hoc network, this extra level of indirection permits building better multicast trees, as it offers more connection alternatives to joining nodes. The resulting tree has less relay nodes, and the average degree of internal members remains reasonably low.

The multicast algorithm aims to rely as much as possible on members for content distribution, thus reducing the number of relay nodes. All the nodes in the network run the multicast algorithm, as for the previously presented DHT, but not all of them are necessarily members of a group. The membership management is decentralized. As a matter of fact, a node can connect to the tree simply by connecting to a member without the source knowing it. This reduces the load on the source and can avoid bottlenecks. Best effort delivery is assumed for multicast messages but additional mechanisms could be easily implemented to support reliable delivery.

3.5 The basic algorithm

To join the multicast tree, a node looks up in the DHT the identifier (key) associated with the desired multicast group (group identifier). Then, the underlying DHT routes the request through the network, traversing multiple nodes toward the source. When receiving a join request, a node checks whether it is a member of the searched group and whether the group has been activated by a multicast message, or not. In case it is a member or its membership has been activated, it replies as a member and proposes itself as a potential parent to the requester while forwarding the request toward the source accordingly to the DHT lookup algorithm. Otherwise, it simply forwards the request to the next hop accordingly to the DHT lookup protocol. Indeed, if a node is not integrated in the tree, it should not propose itself as a parent because this could partition the tree. Finally, the joining node may receive several proposals from potential parents, with at least one in the worst case, because all the requests should reach the source.

To join the tree as soon as possible, the requester accepts the first connection proposal it receives. The upcoming proposals from potential parents will all be considered and the requester will accept another proposal only if (1) it has not yet received a multicast from its parent; (2a) the distance to the new parent

is shorter than to the old parent, and the new distance to the root (i.e., the source) is no more than twice as much as the old one; or (2b) the new parent is at the same distance as the old one, but the distance to the root has shortened. At the end of the process, the requester is connected to the tree with the source or the member that it considers as being the best parent among all the received proposals.

With this connection procedure, many members get linked to the source with direct paths including many relay nodes. Since this situation is clearly not optimal, a number of mechanisms have been developed to improve the tree structure and to reduce the number of relay nodes involved. The first one is used during the lookup phase and tries to provide more potential parents, while, the second and third one rely on information added by nodes to the multicast messages for reorganizing the tree. Finally, the last one gathers broadcast message information to find potential children. These extensions exploit MANETs' particularities to improve the tree structure at low cost and are always cumulated when applied. (i.e., extension n also incorporates extensions $m < n$).

3.6 Finding more potential parents

The first improvement of the algorithm aims at taking advantage of a specificity of MANETs, namely the broadcast mode of communication. Indeed, all the nodes within the communication range of an emitting node can listen to the transmitted messages. So, nodes can freely gather information from messages traveling next to them. Basically, the improvement exploiting the broadcast mode characteristics works like the connection method. If a node listens to a join request for a group it is a member of, and this group has been activated by the reception of at least one multicast message, it proposes itself as a potential parent to the requester.

The requester can accept such connection proposals as long as (1) it has not received its first multicast message; (2) it has no children. Indeed, as soon as it integrates into the tree, the risk of tree partitioning becomes too high and as already said, these situations have to be avoided. If the proposal can be accepted considering the previous criteria, the node will connect to the proposed parent if (1) the distance to the new parent is shorter than to the old parent, and the new distance to the root is no longer than twice the old one; or (2) the new parent is at the same distance as the old one, but the distance to the root has shortened.

This improvement is interesting as no extra messages are generated and listening to communication is typically a cheap operation, yet it often allows improving the structure of the tree. In particular, listening to messages avoids pathological situations where two multi-hop requests cross but do not traverse a common node.

3.7 Finding better parents

The second improvement exploits another particularity of MANETs, the multi-hop routing of messages. Indeed, a multicast message between a parent and a child may traverse several relays. Different messages may then have common relays along their paths, as illustrated by the gray circle labeled 1 in Figure 3.3. In fact, if two messages follow the same physical path, their destinations are likely to be in the same area of the network.

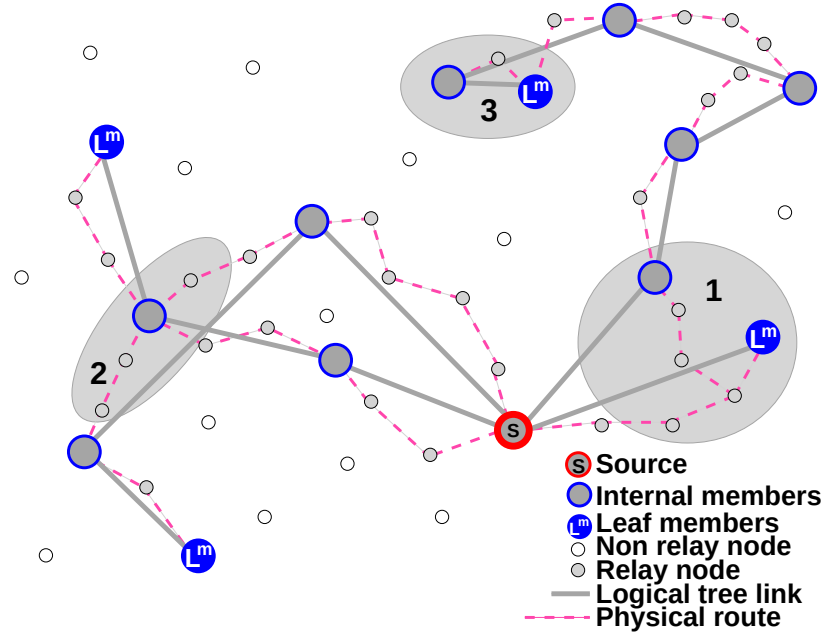


Figure 3.3: Sample scenarios illustrating a tree not organized optimally.

Thus, a node can inform one of the members that the other one may possibly be a better parent or child in the multicast tree. When a relay node forwards a multicast message, it memorizes the group, the message identifier and the destination address (the group and the message identifier uniquely identify the message). If a relay node receives another copy of the same multicast message (same group and identifier), it memorizes the new destination and adds the previously memorized address to the message before forwarding it. If the messages have more than one common relay, only the information added by the last one is kept in the message.

A member checks if an address has been added when receiving a multicast message. If so, it sends a message to the other member and proposes to become its parent. A reconfiguration takes place only if (1) the new parent is closer than the current parent, (2) the new parent is not a descendant of the new child in the multicast tree and (3) the node is outside of the current branch.

This validity check avoids partitioning the tree and losing connection with the source. Moreover, it is important that members in such procedures check that the last multicast identifier they received is the same and the most recent one for both. This guarantees that they consider the same network state.

This optimization is interesting because by shortening the distance between parents and children in the tree, it also reduces the network load. Moreover, the proposed solution requires no additional messages and the space overhead in the multicast message to transport the address of the proposed member is negligible. Figure 3.4 shows the effect of this improvement on the tree presented in Figure 3.3.

3.8 Removing redundant parents

This extension considers, like the previous one, the multi-hop connections between members. In particular, it tries to solve pathological problems where a member is also a relay node or a member is on a multi-hop path to its parent.

It may happen that a node is both, a member and a relay on a multi-hop path between a parent and its child in the same logical tree as presented by the gray circle labeled 2 in Figure 3.3. Manifestly, the resulting structure is not optimal and this situation must be avoided as the affected node receives the same message twice.

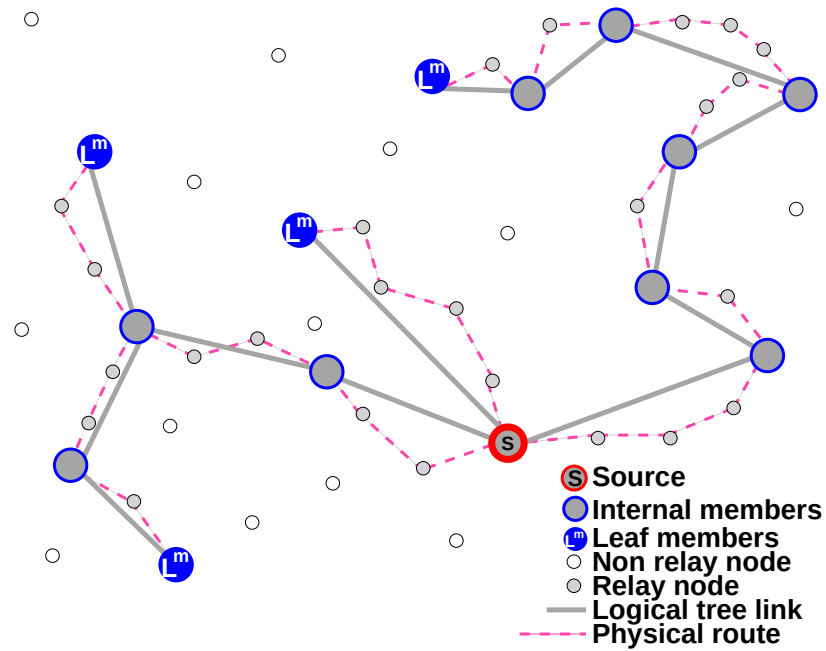


Figure 3.4: Scenarios of Figure 3.3 after applying optimizations.

Consequently, a member that receives the same multicast message from different nodes keeps only the connection with the physically closest internal member (i.e., its parent) and discards other paths. It also promotes itself as the new parent of the addressee of multicast messages that were only transiting through it.

If a member is on a multi-hop path to its parent, as presented by the gray circle labeled 3 in Figure 3.3, the concerned member disconnects from its former parent and adds that node to its own children. Then it becomes the child of its former parent's parent, thus possibly promoting itself as a new inner member of the tree along that path (if it was previously a leaf member).

In either case, one physical path is discarded or rearranged, and the number of relay nodes is reduced, as demonstrated in Figure 3.4 relatively to Figure 3.3. Moreover, the resulting tree better maps to the underlying topology and the multicast efficiency is improved.

3.9 Overhearing multicast messages

Finally, the last extension takes again advantage of the broadcast communication. Indeed, during their distribution, multicast messages may travel on paths physically close to members without reaching them. As messages can be broadcast at all times, originated from each of its neighbor, a node must always be aware of the different communications occurring in its neighborhood. Otherwise, it may miss some of its messages. Usually, messages received by a node, but not intended to it, are simply discarded. Here, on the contrary, members may keep these messages and memorize their content for future use.

The source can then decide to intermittently allow the members from a selected branch of the tree to use the collected information by setting a flag in the multicast message sent to this branch. Each member of this branch then searches in the subset of valid encountered multicast messages the one having the largest distance with its sender in terms of physical steps. A message encountered in the neighborhood is considered as valid if (1) it has the same time stamp as the last multicast message received by the current member; (2) the current member has never considered the message's destination as the worst case and; (3a) its destination is outside the current branch or; (3b) the destination is a descendant of the current member. Finally, the current member proposes to the addressee of the worst message found to become its new parent.

As introduced previously, the source allows a branch only to intermittently use this method and not continuously. This way, the tree structure can stabilize and, when there are opportunities for improvement, the algorithm tries to provide fresh enhancement opportunities by solving the known worst case using the gathered information. Moreover, by opting for the worst unhandled case only, a single message to a potential child is necessary, thus limiting the network load and avoiding several retries of unsuitable solutions.

As reconnection proposals can be sent to almost all members of the tree, the risk to break the tree is not negligible. The constraints described above are necessary and sufficient to avoid breaking the tree, because they always forbid a member to become the parent of a previous one. Time stamps are used to ensure that the situation considered for reconfiguration is up to date and not an inconsistent deprecated one.

3.10 Evaluation methodology

The multicast tree construction algorithm has been evaluated on top of the experimental system used also to evaluate the DHT. A supplementary layer has been added on top of the existing ones (routing layer and DHT layer). These different layers communicate with each other through dedicated methods. We shortly recall the configuration of the simulation system here. For more details, please refer to the Evaluation methodology in section 2.8.1.

3.10.1 Experimental environment

As previously, we considered a set of simple scenarios to evaluate the system. We present the results issued from static scenarios, where the nodes are randomly distributed in a rectangular area according to a uniform distribution and keep their position throughout the whole simulation. In addition, we have also evaluated mobile situations, where the nodes are distributed uniformly in a square area similarly to the static scenario, but where they can randomly decide to move or to stay for a randomly chosen period of time t . As a node decides to move, it picks up a random speed s in the interval $0 \leq s \leq 2m/s$ and a random direction. Then it moves in the chosen direction during time t at speed s . At the end of the period t , a node follows the same procedure and decides whether to stay or to move on. If a node, while moving, reaches the border of the simulation space, its direction is altered to continue its displacement inside the rectangular space. The edge of the Cartesian area is computed dynamically with the regard of the desired density. In our simulation configurations, the edge is 1414 meters for the 1000 nodes network and 3162 meters for the 5000 nodes network.

Two nodes are connected with each other and can communicate, if they are in the communication range of each other. The communication range is defined by a given radius. We ensure that the nodes form a connected graph.

Similarly to the DHT experiments presented in Chapter 2, the DHT identifiers are randomly assigned to nodes and a multicast group identifier is randomly selected within the logical space of the DHT. At the beginning of the simulation, we warm up the DHT by performing 100 lookups to populate the DHT routing tables (for more details, please refer to Section 2.8.1). After the warm up phase, one to three randomly selected nodes issue a request for the DHT with the defined multicast group id as key. This process continues until the desired number of members is reached. At the same time, the distribution of multicast messages starts. Every one to five simulation steps, a message is sent through the tree. The simulation stops once the tree structure stabilizes (i.e., when the algorithm stops producing changes to the tree structure for static scenarios and after a given duration following the time the last node joined the tree for mobile situations).

We experimented with different configurations and network sizes:

- Network sizes: 1,000; 5,000;
- Communication range: 100 meters;
- Simulation space edge: 1414 meters(1K nodes), 3162 meters(5K nodes);
- Connectivity: the average number of physical connections of a node is (network density) varies between 13 and 16;
- Multicast group members: unless specified, 10% of all the nodes join the tree;
- Mobility: randomly chosen speed in the interval $0 \leq s \leq 2m/s$ in random directions during a finite time.

Using these configurations, we have evaluated the different versions of the multicast tree construction algorithm:

- Basic: the construction algorithm with no extensions;
- Ext. #1: during join, members listen to messages and propose themselves as parent when applicable;
- Ext. #2: when sending messages, we try to identify common sub-paths and reconnect members to better parents;
- Ext. #3: we prevent nodes to receive duplicate messages when acting both as member and relay node in a multicast tree;
- Ext. #4: a member listens to multicast messages to propose becoming the new parent of another member that is the farthest away from its parent.

When applied, these different extensions are always cumulated in the presented results.

This multicast algorithm has been compared to the unicast (unicast) and a shortest path tree (SPT) message distribution.

The unicast message distribution is done directly by the source controlling the information distribution and the membership. The source sends the content directly to each of the interested nodes independently. As the source is responsible for the membership management, the nodes interested in the provided content directly connect to the source.

In the shortest path tree, the information distribution and the membership are centralized and controlled exclusively by the source. The interested members have to directly subscribe to the group by contacting it. To distribute the information, the source transmits the message through the shortest physical path to all the registered members. Then the messages are grouped if they have

a common next hop, thus forming a shortest path tree closely integrated with the physical network and able to efficiently adapt to the topology changes.

In the following results, we evaluate the structure of the tree generated by the multicast algorithm. In particular, we consider the degree of the internal members, the number of relay nodes, the relative distance between two members and the total number of transmissions required for reaching all the members. We also use a normalized multicast cost evaluation proposed by Jaquet and Rodolakis in [Jacquet 2005] to get measurements that do not depend on the size of the network and allow us to compare all the approaches and network sizes with each other. The general multicast cost corresponds to:

$$R(n) = \frac{\text{multicast cost}}{\text{average unicast cost}}$$

Where the multicast cost is the number of hops (=number of transmissions) necessary to reach all the members of the tree and the average unicast cost is the average physical route length from the source to a random member. To compute the average unicast cost in a defined network size, we consider that the members are all directly connected to the source, then we compute the average unicast cost in a number of transmissions (= number of hops) as unicast cost reference.

We have also built the tree without using the DHT to support the source localization. A joining node directly contacts the source for connecting to a multicast tree. We use the algorithm with all the extensions and evaluate the number of relay nodes included in the tree.

Finally, all the presented results are obtained from averaging the results of 10 experiments conducted on different node distributions with different DHT key distributions and different multicast group keys.

The validity of the experiments is discussed at the end of Section 3.11. We base this evaluation on the coefficient of variability (COV), which represents the ratio of the standard deviation to the mean. It is an interesting measurement because it allows scale-free comparison from data series to another, even if the means are drastically different. This is basically not the case when using the variance for this purpose.

3.11 Experimental results

As introduced in the previous section, there is an interesting fact to notice about the use of the DHT to support the multicast algorithm. Although the DHT lookups efficiently locate the source, the requests use a physical path longer than the one provided by the underlying routing algorithm. However, as shown in Table 3.2, the tree built with the DHT includes systemically less relay nodes. In fact, the small indirection of the DHT routing increases the probability to locate a better parent to connect to the tree.

Network size (# of nodes)	500	1,000	2,500
No-DHT	12.5%	13.82%	15.64%
DHT	10.46%	11.32%	11.69%

Table 3.2: Percentage of relay nodes involved in multicast trees with direct request or DHT lookup.

3.11.1 Degree of member nodes

Figure 3.5 presents the cumulative percentage of members in message relaying relatively to their degree. The results also compare the mobile and static scenarios. The ideal case for a good tree structure is a curve with a Y value growing very fast and stopping with a small X. Indeed, such a curve means that many members participate in the message distribution, but their degree is limited.

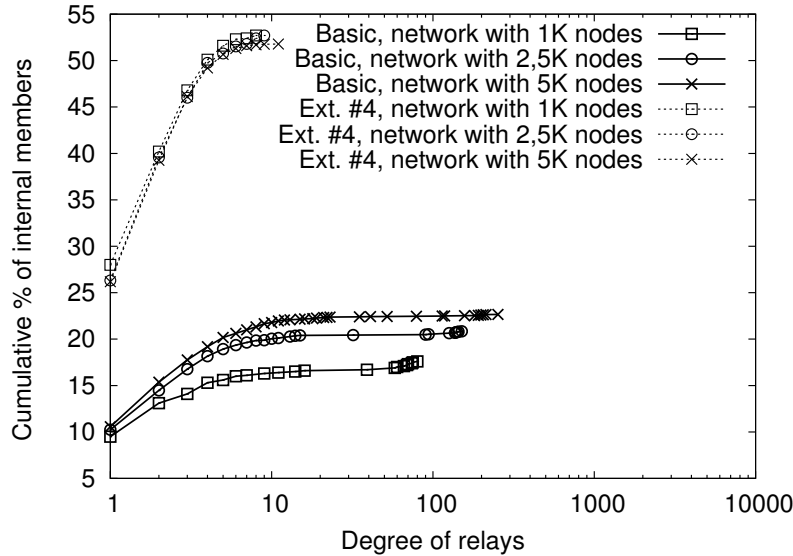


Figure 3.5: Cumulative percentage of members acting as internal members as a function of their degree (static - mobile comparison).

Clearly, in the tree produced by the basic algorithm for both static and mobile configurations, less than 30% of the members participate in the message distribution and 60% are then only leaf members. Moreover, some nodes have quite a high degree (most notably the source). In contrast, when the different extensions have been applied, more than 50% of the members help to distribute the content,

thus acting as internal members. The degree of the internal members does not exceed 7 for all the considered network sizes. Hence, the algorithm contributes to the distribution of the multicast and membership management load between the members. Comparing the mobile and static scenarios, we can observe that the algorithm does not suffer from the nodes' mobility.

3.11.2 Relay nodes

As a result of the increased number of internal members relaying messages with the improved algorithm, the number of relay nodes should decrease. This is confirmed in Figure 3.6 for all the versions of the multicast algorithm except for the basic version. Clearly, the shortest path algorithm and unicast distribution methods use less relay nodes than the basic algorithm. Indeed, as same routes are used many times to reach the members, this limits the number of relay nodes implicated. By using members to help in the message distribution, the multicast algorithm creates branches not connected to the source, thus including additional relay nodes. At the same time, as demonstrated by Figure 3.11, the basic multicast algorithm is close to the unicast and shortest path tree as an important number of members are directly connected to the source. Consequently, it uses approximatively the same branches around the source like the unicast and the shortest path tree, but has additional branches because of the presence of internal relays.

As the multicast algorithm uses more internal members to distribute the messages, it reduces the load of the source. This is confirmed by the number of transmissions presented in Figure 3.8. The transmission cost of the basic algorithm is only slightly higher than the transmission cost of the shortest path algorithm and noticeably lower than the transmission cost of the unicast distribution method.

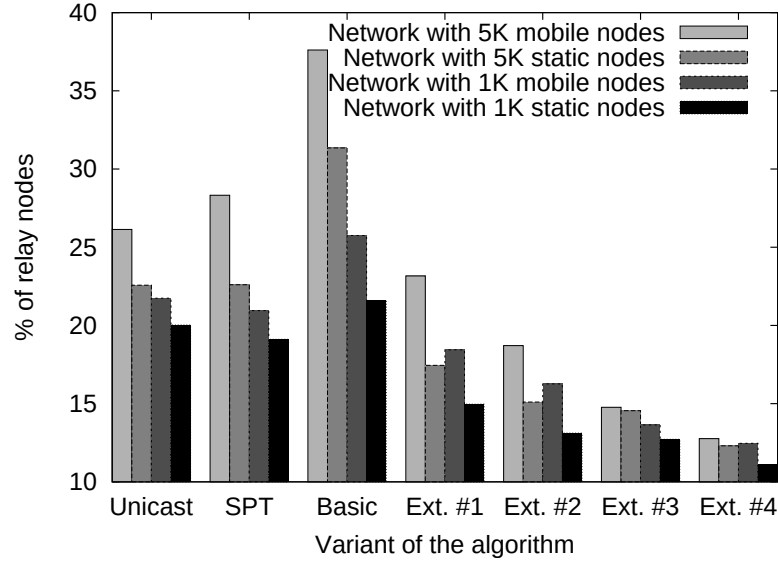


Figure 3.6: Percentage of relay nodes involved (static - mobile comparison).

Although the basic algorithm requires more nodes in the mobile context, the trees built with all extensions in the mobile scenarios need roughly the same number of nodes as in the static experiments. This result is really interesting. Recall that the simulation continues after the tree has been built. For the static situation, it allows to stabilize the tree structure. For the mobile situation, as the nodes continue to move and as we evaluate the tree at the end of the simulation only, it demonstrates that the algorithm and its extensions not only build efficient multicast trees, but are also able to maintain their good properties. This fact is confirmed by other measurements presented in Figure 3.5 and Figure 3.7.

3.11.3 Average distance between member nodes

The multicast algorithm builds a tree in which members that are physically close can discover each other and connect by just passively gathering information as messages are broadcast. Thus, the physical path length between two consecutive members is small, as demonstrated by Figure 3.7, and is in all cases obviously smaller than the one in the shortest path tree and the unicast solutions. This is not surprising as all the members are leaf members directly connected to the source. Consequently, the average distance between two members (i.e., the average distance to the source) is maximal, since no connection to a physically close member is envisaged.

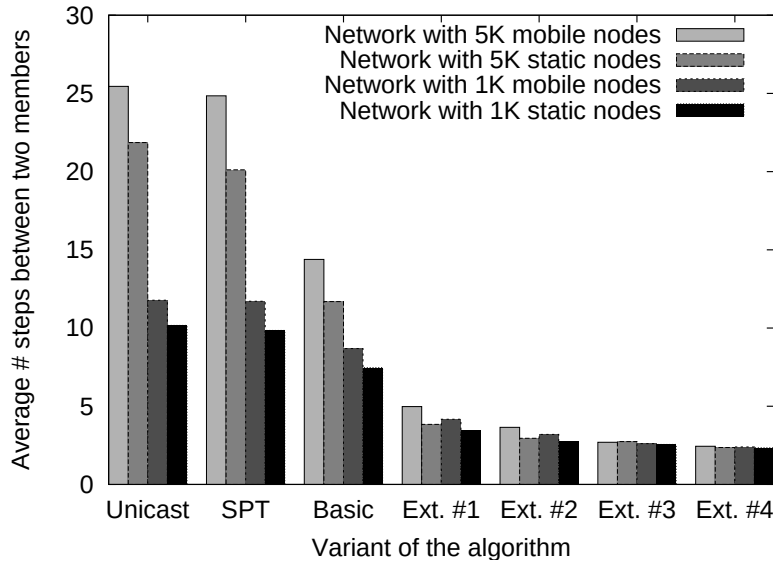


Figure 3.7: Average number of physical steps separating two members (static - mobile comparison).

The tree built by the presented algorithm connects close members together, enabling it to be closely mapped to the topology of the underlying network. This is

an advantage in MANETs, as short paths reduce the probability that the route will break, and when this happens the time to rebuild it is minimal.

3.11.4 Transmissions required for reaching all members

Since this algorithm is specifically adapted for MANETs with resource limited mobile devices, the total energy consumption is an important factor. If the implicated nodes use too much energy when they communicate among themselves, the network lifetime is affected. Therefore it is crucial to avoid unnecessary energy consumption. This can be achieved by limiting the total number of transmissions required to cover the tree, because each wireless communication requires energy. As shown in Figure 3.8, the multicast algorithm needs less transmissions than the shortest path tree and the unicast distribution. Only the basic version requires a few more transmissions than the shortest path algorithm. Indeed, as already explained, both trees have the same structure around the source, but the basic algorithm builds more branches between the members. The multicast algorithm with all extensions saves energy by reducing the required transmissions to cover the whole tree.

3.11.5 Multicast cost

The multicast cost of the basic multicast algorithm is smaller than the multicast cost of the unicast distribution method but slightly higher than the shortest path tree as shown in Figure 3.9. As previously explained, the basic algorithm builds a tree with a structure close to the shortest path method with internal members. Thus some branches are created. A few additional transmissions are required to cover these branches, therefore increasing the multicast cost. The multicast algorithm is cheaper than the shortest path tree and the unicast in all other cases.

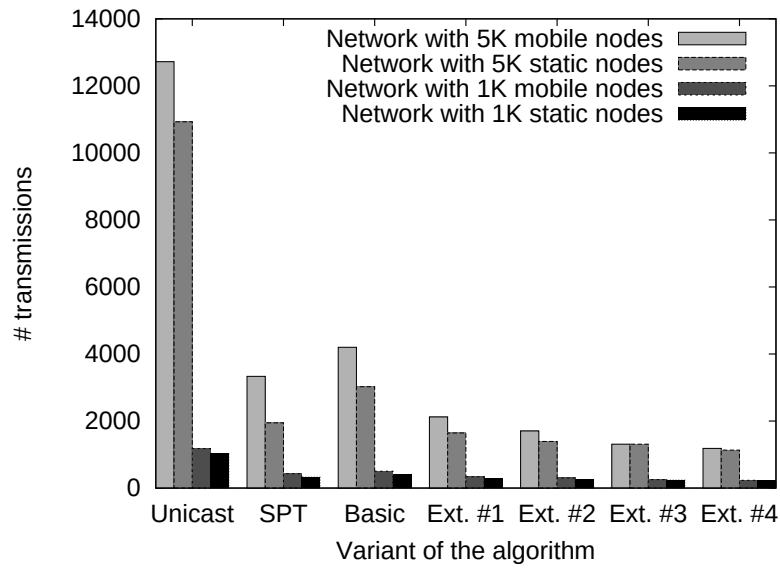


Figure 3.8: Average number of transmissions required to cover the tree (static - mobile comparison).

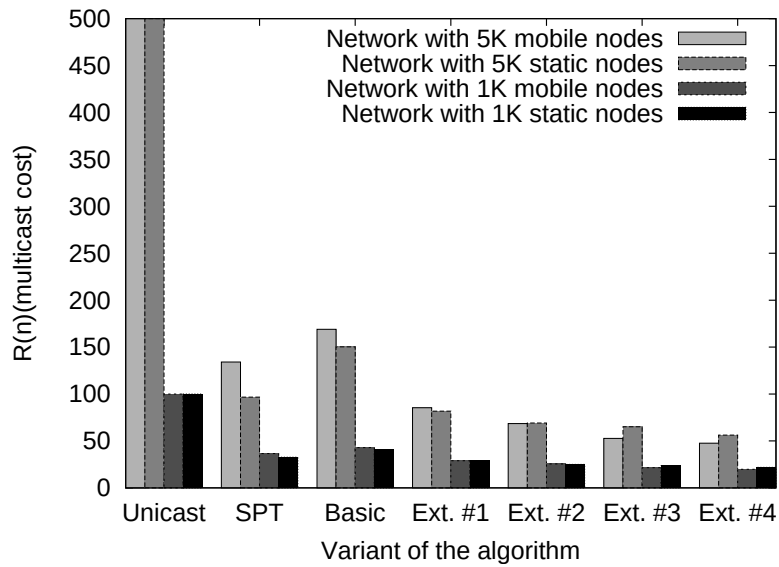


Figure 3.9: Average multicast cost ($R(n)$) (static mobile comparison).

3.11.6 Scalability

When applying the four extensions, the physical distance between two members, the number of involved relay nodes and the degree of internal members are nearly the same for all the considered network sizes. At the same time, the experiments in mobile conditions demonstrate that the algorithm is able to maintain the tree structure independently of the number of nodes in the network.

These two results support the claim for the scalability of our approach. This is confirmed by Figure 3.10, where the multicast cost of the improved algorithm is not only lower but also grows slower than the multicast cost of the shortest path tree and unicast.

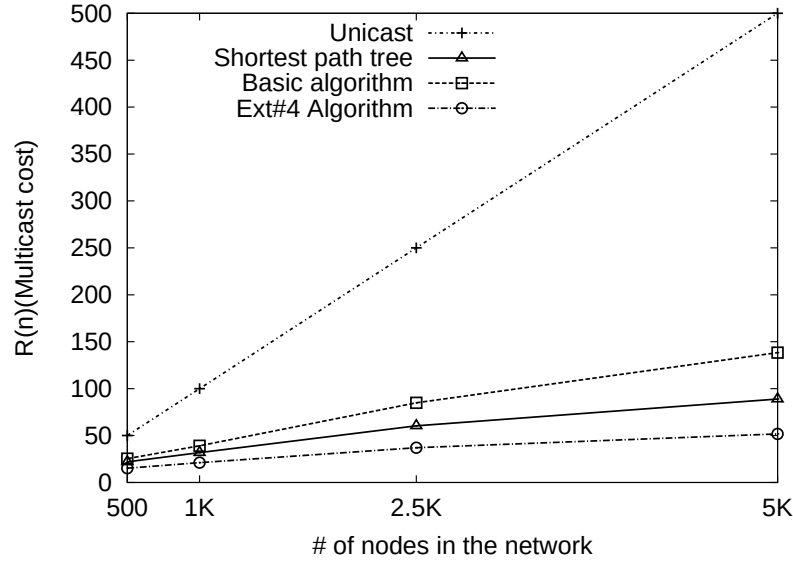


Figure 3.10: Multicast cost evolution in different network sizes (static configuration).

3.11.7 Variability of the results

For several experiments presented here we have evaluated the coefficient of variability (COV). It represents the ratio of the standard deviation to the mean. This measurement allows scale-free comparisons out of variability consideration, as opposed to the variance.

The average number of steps between two members (Figure 3.7) has a maximum COV of 0.7% for all network sizes and configurations with the extended algorithm. This means that there are no significant differences between the conducted experiments. Similarly, considering the number of relay nodes (Figure 3.6), we obtain a maximum COV of 8%. Only the number of transmissions varies noticeably. Indeed, the routing algorithm implementation provides some optimizations, like the grouping of messages, whose applicability depends on the topology of the ad-hoc network and the position of the member nodes. This improvement increases the variation of the number of transmissions required to cover the tree.

3.11.8 Structure of the tree

Finally, to illustrate the effect of the improvements on the tree structure, Figure 3.11 shows a tree built by the basic algorithm. Clearly the structure is not optimal and only a few members participate in message relaying. This tree can be compared to the tree in Figure 3.12, which corresponds exactly to the same network configuration and the same source, but with the improved algorithm. The structure is obviously much better adapted to the physical topology. Moreover, the load on the source is greatly reduced and many more members participate as internal members.

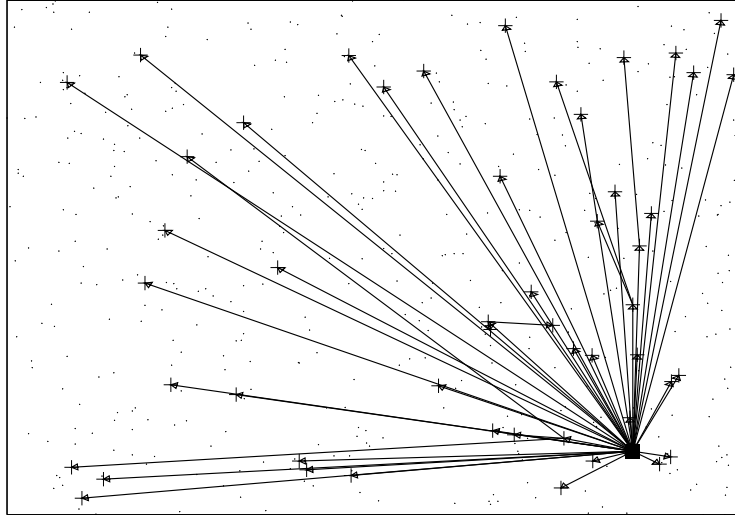


Figure 3.11: Sample tree built by the basic algorithm (static configuration).

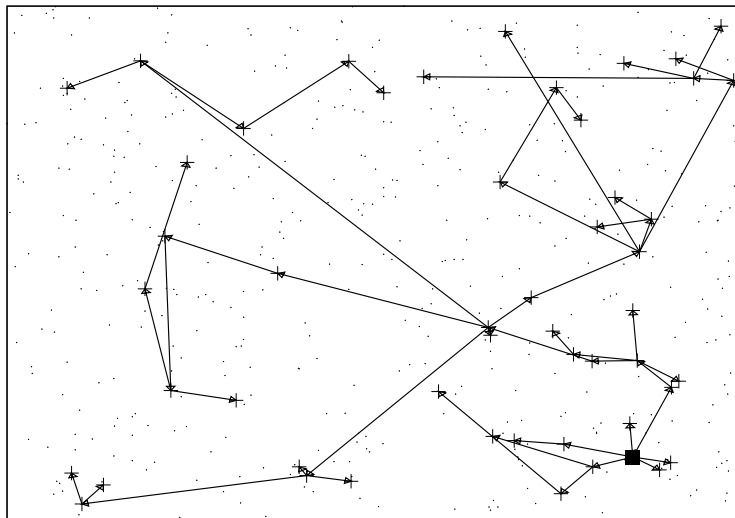


Figure 3.12: Sample tree built by the improved algorithm (static configuration.)

3.12 Summary and Discussion

Similarly to the information lookup (or retrieval) algorithm developed in Chapter 2, information distribution can be realized through flooding-based approaches. In this chapter, we presented an algorithm for the construction of efficient multicast trees using an underlying ad-hoc DHT overlay. Our algorithm strives to create trees that involve as few relay nodes as possible, requiring a limited amount of transmissions, with short inter-members paths and good scalability. Simulation results indicate that our algorithm meets these objectives in the considered network settings and is able to maintain a good structure in a mobile environment. This approach also achieves reducing the load on the nodes, while successfully distributing the membership management and sharing the load of information distribution among participating devices.

In this chapter and the previous one, we presented our algorithms for lookup and multicast, and a set of simulations and emulations to validate their performances. In the following chapter, we present an implementation of both algorithms on Windows Mobile powered PDAs connected through standard WiFi channels. The implementation was done in line with the simulation model with only little adaptations to the environmental constraints. The architectural design and the details of this implementation are presented in the next chapter.

The Adcast software

4.1 Introduction

Chapters 2 and 3 presented our algorithms for lookup and multicast as well as their evaluations through simulation. The obtained results demonstrated that they may work efficiently on MANETs.

The DHT algorithm has also been implemented in the Freemote emulator on top of another routing algorithm (NST-AODV). The results obtained in this environment confirmed the ones obtained in simulations, except for minor differences due to the implementation choices.

So far, the DHT and the multicast tree building algorithm have never worked together in a real environment. This motivated their implementation on Windows mobile powered PDA. This choice has been driven by the availability of the devices, their diversity and number.

As all the devices are Windows mobile powered and no suitable version of Java was available on them, we had to choose one of the adapted development languages provided by the Visual Studio environment. C# can be compiled just once for all PDA architectures as it is an interpreted language. Being close to the wide-spread Java language, it contributes to support the validity of implementation. Finally, as the number of available devices was restricted (only 6 PDAs were available), the possibility to run the implemented software on Windows powered computers was instrumental for larger experiments .

This chapter details the software implementing the DHT algorithm and multicast tree building algorithm for the PDA/PC infrastructure. The different adaptations of the algorithms required by the environment and their operations are detailed. Both algorithms have been implemented together before in a discrete step simulator and cannot simply be transported to the real world without modifications. In the simulation environment, the position of the nodes was defined by the system and their number did not vary. Here, we developed a simple connection algorithm as it is required that nodes (i.e., software instances) may be able to join the DHT overlay whenever required. No disconnection or churn handling method has been developed as the purpose of this implementation is to demonstrate the feasibility to deploy and run such algorithms on real MANETs devices.

The implementation has been tested on a limited MANET constituted by 6 PDAs and in a larger experiment in a lab of computers simulating the ad-hoc network operations. A difficulty is to experiment different network topologies with so few devices. Thus, a topology manager has been added to the system to test the system's behavior in different network configurations. The obtained results demonstrate that the algorithms developed are suitable for resource limited devices and are suitable for real P2P operations with minor adaptations only.

The remainder of this chapter is organized as follows. Section 4.2 presents the software architecture and its features. We then detail the algorithms' specific implementations in Section 4.3. Section 4.4 presents the evaluation methodology and the experimentation results. Finally, Section 4.5 concludes this chapter.

4.2 Software architecture and features

As in the case of the simulation environment described in Section 2.8.1 or the Freemote emulation presented in Section 2.8.3, the Adcast software has been structured in multiple interacting independent layers (see Figure 4.5). Consequently, the layers' implementations are not fixed and can easily be modified in future versions of the software (i.e., future experiments). Each application layer (or algorithmic layer) can be started independently through the Adcast main window. The main window shown in Figure 4.1 provides the general configuration parameters and assures the integrity of the operations. Indeed, an operation is allowed only if all the previously required ones have been executed (i.e., configuration or layer starting).

The software has many other operation and visualization windows. In particular each layer has its own visualization and command window which is activated as soon as the layer has been started. Figure 4.2 shows the routing layer status window. This window displays information on the current operations of the routing layer. The DHT command window shown in Figure 4.3 permits to start a DHT lookup and gives some basic information on the DHT overlay structure (i.e., logical neighborhood, local DHT key, etc.). The multicast layer information is displayed in a window presented in Figure 4.4. Some useful events are exposed in a log window and facilities are provided to join a multicast group.

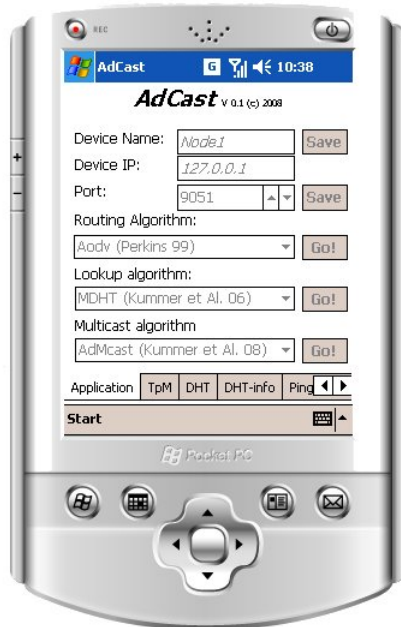


Figure 4.1: Adcast start window (i.e., general configuration)

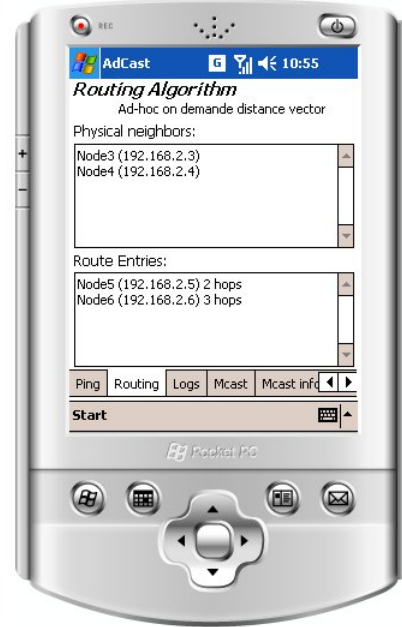


Figure 4.2: Adcast routing window

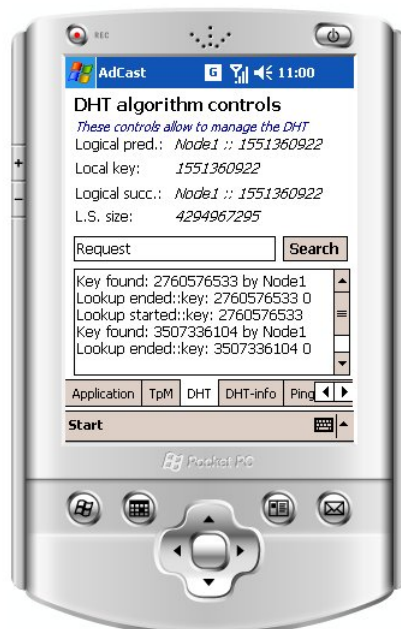


Figure 4.3: Adcast DHT window



Figure 4.4: Adcast multicast window

In the following, we introduce the application's technical architecture under two different perspectives. On one hand we consider the role of the different layers. On the other hand, we present the different interactions among the layers.

The general software architecture is presented in Figure 4.5 and the role of each layer is shortly summarized. The role and interactions of each layer is then detailed in the following sections.

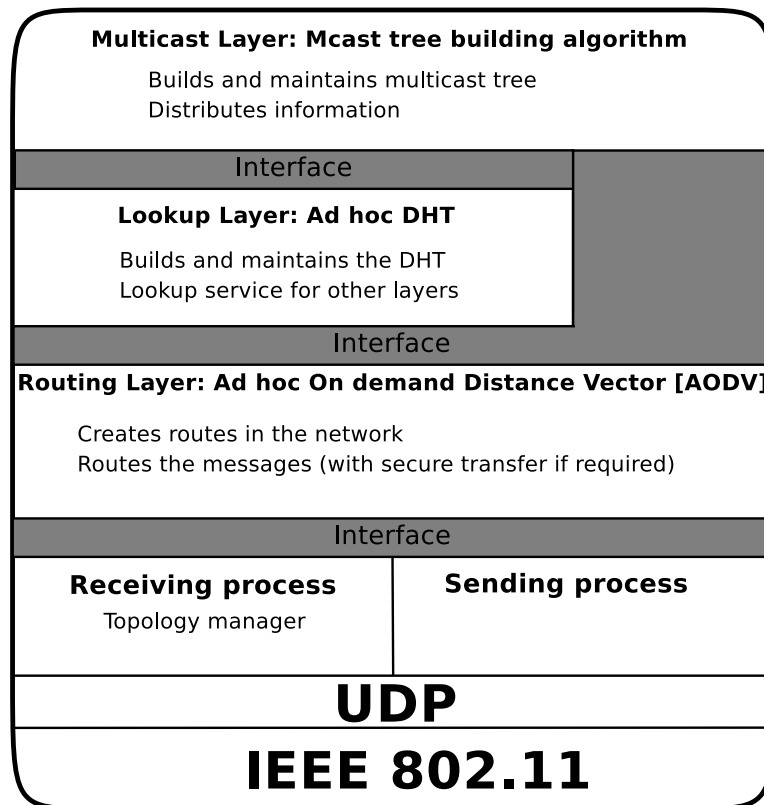


Figure 4.5: Layers structure and roles

4.2.1 Sending and receiving layer

The lowest layer is dedicated to sending and receiving messages. It is linked with the used network protocol, in our case, UDP. This layer is divided in two independent processes as shown in Figure 4.5 :

- The sending process
- The receiving process

They receive and send the messages in methods that start the operation and give the control to another thread. Thus, if an operation fails, the global operation can continue as if nothing happened. It is important here, for reliability purpose, to avoid that transmission errors affect the whole software operations.

The receiving process is linked with a topology management tool (the topology manager). As a matter of fact it is really difficult to build an ad hoc topology. First a vast area is required to scatter the devices so that they will not be connected all together. Secondly, considering that we have only few devices at our disposal, they have to be close enough to communicate together. Considering this, it is quite impossible to arrange the devices to have particular topologies and to reliably conduct experiments. The topology manager allows us to set up the desired topology with all the devices operating in the same limited space (e.g., an office). In addition, due to the topology setup facility, the Adcast software can be easily deployed in a lab of desktop computers for experiments. The messages arriving from devices set to “out of the communication range” are discarded. They never reach the upper layers. The topology can be modified at any time, therefore simulating nodes movements if required.

The sending process first serializes the messages received from the routing layer into a bit array, then sends them on the network, and the receiving process deserializes all the received and overheard messages and transfers them to the routing layer.

4.2.2 Routing layer

The routing layer operates directly on top of the sending and receiving layer. Its role is to build, to maintain and, if necessary, to discard the routes in the MANETs. The layer is responsible of delivering the messages to their destination by sending the local messages or forwarding the ones in transit. Two kinds of messages are considered here, the messages with a defined destination and the broadcast messages. The layer also provides an interrupted routing mode. In this mode, all the messages in transit through a node are transferred to the upper layers. Then, the algorithms running in these layers may either alter the messages, change their destination, discard them or simply forward them as if no interruption happened. In the following, we name this kind of messages “interrupted messages”. Finally, as we made no assumption on the underlaying network protocol, the routing layer provides a secure routing mode with end-to-end transmission control. Typical MANETs experience high message loss. In some scenarios, a simple periodical message replay is enough while in others the sent messages have to reach their destination. Thus, it is important to dispose of an end-to-end transmission control mechanism.

The various interactions between the routing and the other layers are represented in Figure 4.6. The routing layer receives the message objects from the sending and receiving layers. The differentiation between the received and the overheard messages is done by the layer below. Then the routing layer can handle both kinds of messages as desired. In the sending process, the routing layer provides the message objects to the sending and receiving layers with all the routing information included.

The routing layer delivers the overheard, interrupted (if required by the layer) and received messages to the upper layers. These layers send all their messages through the routing layer. They have to include the final destination or the

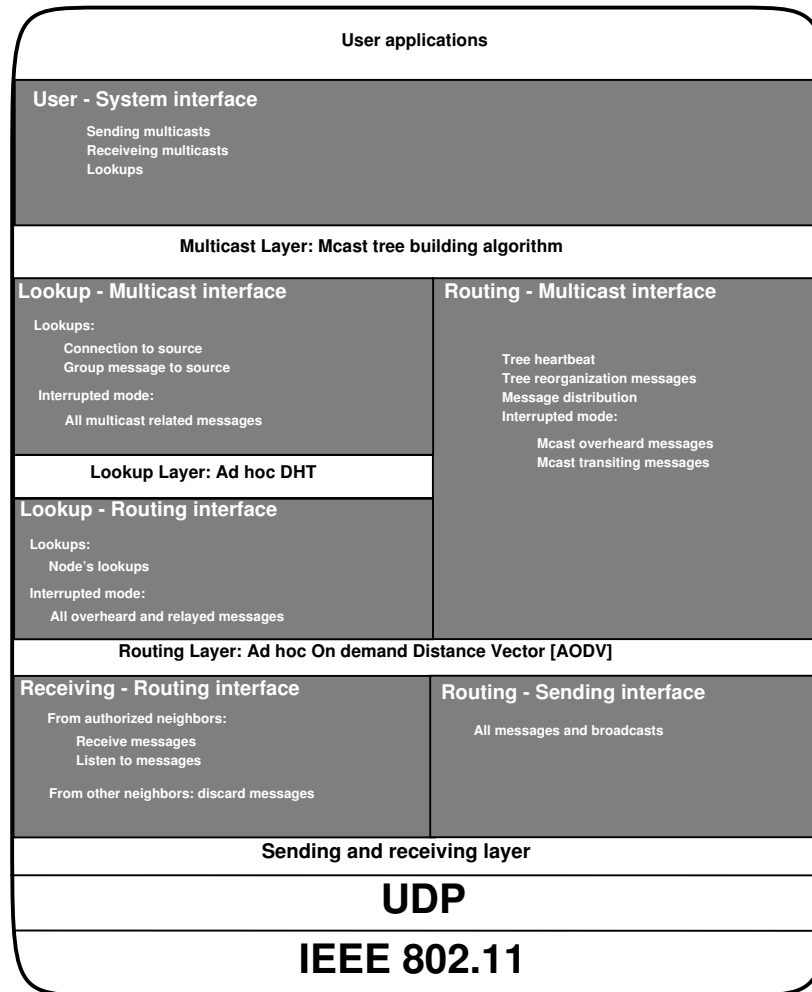


Figure 4.6: Interactions between layers

broadcast address and set up the desired routing options (i.e., interrupted routing, secure routing, ...). The routing layer does the rest of the work and fills the required fields for the transmission.

In Adcast, the routing layer implements AODV [Perkins 1999]. The specific implementation is described in Section 4.3.1.

4.2.3 Lookup layer

The lookup layer is responsible for building and maintaining the logical lookup overlay. It provides a lookup service to the user and the other parts of the software. A lookup may either fail or succeed. When a lookup succeeds, the following events happen:

- The node responsible for the key sends a success notification to the requester. If required, it informs a local process (i.e., another layer) with the incoming request.
- When receiving the success notification, the requester's lookup process notifies the local requesting process with the network address of the node responsible for the key (the network address is here composed by the device's name and the device's IP address).

In case of a lookup failure, the local lookup layer notifies the requesting process.

As for the routing layer, the lookup layer provides an interrupted lookup mode where the lookups are passed to the requesting layer at all steps. This mode must be set up when calling the lookup service. The lookup layer may also be asked to notify the upper layer when a request reaches the node responsible for the desired key. This mode, again, must be setup when calling the lookup service.

For better understanding, the interactions between the lookup layer and other layers are shown in Figure 4.6. The lookup service can be called through a dedicated interface by any local process in the application. The calling process must pass the lookup key, set the desired options (i.e., interrupted routing, success behavior), and may provide some additional information for particular success behavior.

All the messages from the lookup layers are sent through the routing layer. The lookup layer uses three types of messages:

- *Connection messages* : These messages are designed to support the connection process. They use the end-to-end transmission control (secure routing mode) provided by the routing layer.
- *Request messages* : They carry the lookups through the networks. These messages use the interrupted routing mode provided by the routing layer.
- *Heartbeats* : These are small messages used to control the logical structure state (i.e., if the logical neighbors are still in the network).

The lookup layer receives all messages from the routing layer (i.e., its own and overheard messages). In Adcast, this layer implements the algorithm presented in Chapter 2. The details of the implementation are presented in Section 4.3.2

4.2.4 Multicast layer

The multicast layer builds and maintains the multicast overlay. It also manages the multicast membership. Its two functionalities are:

- Connect to a multicast group
- Send a message to a multicast group

For both tasks, it relies on the lookup layer to deliver either the connection requests or the multicast group messages to the node responsible for the desired multicast group. Then, from the node responsible for a multicast group to the members, the messages are passed directly to the routing layer. These functionalities are available to the user only and cannot be called by another process in this implementation.

The multicast layer interacts with the routing layer and the lookup layer depending of its needs and the message type. Figure 4.6 shows these interactions. They are detailed here in relation to the message type:

- *Group connection request* : Message sent through the lookup service. It requires the interrupted lookup mode and the particular success behavior and may transport additional data if required. The particular success behavior forces the lookup layer to provide lookup messages to the multicast layer at the node responsible for the desired key.
- *Group multicast request* : Message sent through the lookup service. It requires the same particular success behavior as the Group connection request and transports the group message as additional data.
- *Group multicast message* : Message sent through the routing layer. It requires interrupted mode and delivers the multicast message to the group members.
- *Reorganization message* : Message sent through the routing layer. It requires end-to-end transmission control (secure routing). This is a service message essentially used for member reorganization.
- *Heartbeats* : Small messages used to maintain the multicast distribution structure alive.

Except for the group connection request and the group multicast request, the multicast layer receives all other messages from the routing layer (i.e., its own and overheard messages).

In Adcast, this layer implements the algorithm presented in Chapter 3. The details of this implementation are presented in Section 4.3.3

4.3 Algorithms

In the following subsections, we detail the implementation of each layer. We discuss their operating process and the differences with the original versions of the algorithms.

4.3.1 The AODV implementation

We based our implementation of AODV on the paper of Perkins and Royer [Perkins 1999]. The algorithm operates “on demand” and in parallel. It reacts to a message reception or to a call to the message sending interface (call done by an upper process). Additionally, two independent processes intermittently wake up. The first one sends AODV heartbeats (AODV-HB). This process maintains the physical neighborhood information. The second process is responsible for cleaning the information kept for the routing operations (i.e., routing tables entries, memorized tags, ...). Indeed, after a certain time, as the network nodes may be mobile, routes may be not up to date or even not exist anymore and corresponding routing tables entries can therefore not be used anymore. This obsolete information uses unnecessarily the limited memory and has to be regularly removed. Due to the interaction of many parallel processes, all the data structures are synchronized to avoid inconsistent information usage.

On a reception call, the algorithm performs the following operations:

- If** the message is a RREQ (route request) or RREP (route response):
 - Proceed according to the AODV specifications [Perkins 1999].
- Else if** the message is an AODV-HB:
 - Refresh the local neighbor list with the sender's address.
- Else if** the message is a broadcast:
 - Update broadcast data.
 - Deliver the message to the concerned layer if resend limit not reached.
- Else if** current node is the message destination or interrupted routing is required:
 - If end-to-end transmission control activated: acknowledge the message.
 - Update the path length in the message.
 - Deliver the message to the concerned layer.
- Else** forward the message:
 - If end-to-end transmission control activated: acknowledge the message.
 - Update the path length in the message.
 - Call the sending method.

The overheard messages are simply transferred to the appropriate layer.

The sending process proceeds as follows: when a message is transmitted to AODV through the sending interface, the algorithm checks whether a valid route to the desired destination is known and active (i.e., routing entry lifetime has not expired) or not. If the routing entry exists, the routing fields of the message are updated with the information presented in the routing table. The routing table entry is reactivated (i.e., routing entry lifetime extended) and the message is sent on the network. If no route exists, the route discovery process from AODV is executed as described in [Perkins 1999].

This implementation works with two types of device identifiers:

- Device name
- Device IP address

The device names are a simple and efficient way to recognize the devices. It is very helpful for experimentations, for example. The IP address is automatically provided by the Windows mobile system. With this approach, even if a device is shortly disconnected and comes back with another IP, all the algorithms may still work. This pair, device name, IP address may also be used to uniquely identify a device in a network like a MANET where no central DHCP server guarantees a unique IP addresses.

The end-to-end transmission control is done at each routing step (i.e., at each node). The sender sends the message every Δt time as long as the maximum number of retransmissions is not reached. The receiver acknowledges each message (for each transmission and retransmission). For efficiency reasons, a message requiring transmission control is directly forwarded to the next step, even if the confirmation process did not end on the current node. Each message is identified by a unique transmission control tag. The process checks the tag and determines if the message has already been received and forwarded.

The route breaking detection method proposed by Perkins and Royer in [Perkins 1999] has not been implemented. The conducted experiments are done with a static setup. As a consequence, route break should not take place. Consequently, this feature was not required for our experiments.

4.3.2 The DHT implementation

The lookup layer implements the DHT presented in Chapter 2. This layer, like the routing layer, operates “on demand” and in parallel. It reacts to a message reception or to a call to the lookup interface (call done by an upper layer). Additionally, two independent processes intermittently wake up. The first one sends heartbeats (LL-HB). This process spreads out the physical neighbors and NoN logical information and keeps alive the links between direct logical neighbors on the ring (i.e., logical space). The second process is responsible for cleaning the information kept for the lookup operations (i.e., neighbors table, NoN table, cache entries,...). After a certain time, as nodes may move, this information becomes obsolete. Indeed, the neighbors change and thus also the NoN. Out of date entries in neighbor and NoN tables must be removed. Indeed, wrong and old information may disturb the algorithm operations leading to mistakes and unnecessarily usage of the limited memory. Due to the interaction of many parallel processes, all the data structures are synchronized to avoid inconsistent information usage.

The lookup algorithm has been implemented as presented in Chapter 2. Only the logical space management policy has been modified. We adopted the approach used in the Freemote’s implementation presented in Section 2.8.3. The nodes are placed at the beginning of their key spaces rather than in the middle. A node is thus responsible for a key that is not necessarily the closest one. This scheme is used to solve the problem of keys being exactly at the same logical distance from two consecutive nodes (i.e., the same logical distance with the previous and the next node on the logical ring). Therefore, the logical distance computation method has been adapted to fit the new logical space management policy.

For each node, we maintain the connection at all times with a node’s successor $\text{succ}(n)$ and predecessor $\text{pred}(n)$ in the ring (i.e., logical space). The connections and paths are maintained by the LL-HB that are regularly sent

by the dedicated process. To minimize the global load, logical neighborhood information is added to the LL-HB for the first physical step on the path to its destination (i.e., $\text{succ}(n)$ or $\text{pred}(n)$). The physical neighbors from the sender handle the LL-HB in the following way:

If the node is the next step on the path to the logical neighbor:

- Update its local information
- Remove the neighborhood information of the message
- Forward the HB as a normal message.

Else the node is not on this path, it updates its local information and discards the message.

To build the logical ring, a connection method has been developed. This connection algorithm is detailed below. For improved clearness, we will often refer to the pseudo code presented in Algorithm 2.

A connecting node first checks if it has physical neighbors. If not, it creates the DHT (line 3). It becomes its own successor and predecessor. Thus, the DHT logical space is created, the connection succeeds. If the connecting node has some physical neighbors, it tries to connect to an existing DHT supported by these neighbors. The node begins by broadcasting a connection request to all its physical neighbors (line 6) and waits during time t for connection proposals. The connection proposal prop_j related to the largest logical space log_j (line 9) is selected in the set of received proposals Prop . Connecting node n_i gets the DHT id located in the middle of the logical space log_j (line 10). n_i becomes now the successor of node n_j (line 11), n_j 's successor becomes n_i 's successor (line 12). At the end of the connection process the load of the largest logical space is shared among nodes n_i and n_j , with n_i becoming the new successor of n_j . This procedure is restarted maxRetr times in case of failure (line 5). When maxRetr is reached, the node creates the DHT (line 17). It becomes its own successor and predecessor (line 18). The DHT logical space is now created, the connection succeeds.

21: end procedure

This connection procedure principally works locally and one single message needs to be routed through a multi-steps path. This procedure also reduces the size of the logical space managed by the nodes. The messages exchanged during the connection process require the end-to-end transmission control (provided by secure routing). The connection is a sensitive phase, bad operations may compromise the integrity of the logical space.

The lookup service can be called by the user or by an application process (e.g., another layer). The algorithm looks up the DHT and waits for an answer. If no answer is received, the request is replayed. After 2 retries (i.e., 2 minutes), the requester receives a failure notification. If a lookup is initiated by a process and the interrupted mode is activated, the DHT algorithm forwards all overheard and received messages. The interrupted routing mode proposed by the routing layer is activated for all its own messages.

4.3.3 The multicast tree building algorithm implementation

The multicast layer implements the multicast tree building algorithm presented in Chapter 3. In contrast to the algorithm described in the previous sections, the multicast tree building algorithm operations are sequential and are controlled by a single process. The multicast tree building algorithm dispatches state information for its structure optimizations. The order of messages and operations is therefore important. The messages received or overheard by the algorithm are buffered. The process executes the sequence of operations and messages in order of their arrival. Only the connection requests (received and overheard) are handled immediately if the system state allows it. The algorithm as described in Chapter 3 requires no modifications to work in the Adcast software.

The main process of the multicast tree building algorithm works as follows:

Handles the multicast messages:

Extract the messages from the buffer.

Each message is analyzed. The proposed operation is done if all conditions for the demanded operation are fulfilled¹.

Handles the overheard messages:

Extract the overheard messages from the buffer.

Each message is analyzed. The proposed operation is done if all conditions for the demanded operation are fulfilled¹. Otherwise the message is kept for a future optimization process.

Sends the multicast heartbeats (MCAST-HB) through the locally rooted tree:

The MCAST-HB are sent every 10 seconds.

The usage of overheard messages to improve the tree structure for a tree branch is activated during this process (see Section 3.9).

Cleans the system:

Removes all obsolete memorized data. This operation keeps only recent information.

The interrupted lookup mode provided by the lookup layer is always used for the connection requests sent through the DHT. The connection algorithm requires this mode to be fully functional and efficient. The multicast message distribution (i.e., multicast group message and heartbeats distribution) requires the interrupted routing mode provided by the routing layer. Again, the tree structure optimization methods (see Sections 3.7 and 3.8) need this feature to be fully functional.

¹ The operations and their execution conditions are described in Sections 3.5 to 3.9

4.4 Experimentations

The first experimentations of Adcast have been done on a set of six PDAs. All the software layers worked together and it has been possible to:

- Route ping messages from a node to any other one and get the ping answer.
- Connect the DHT running on all the participating nodes and lookup arbitrary keys.
- Build a multicast tree and observe the structure changes.

We do not present statistics obtained from the ad hoc network from PDAs. In fact, it is not possible to obtain significant measurements on such a small network. Moreover, we have no comparison possibility with the simulations presented in Chapters 2 and 3. Hence, we set up Adcast in the experimental environment described in the following section.

4.4.1 Experimental setup

The objective of the experimental setup described hereafter was to obtain comparable configurations for Adcast and the simulation environment described in Section 2.8.1 that is used to evaluate the DHT and multicast tree building algorithm. In this way, we can collect results that are comparable.

We deployed Adcast on a set of 20 Windows XP networked computers. Each one ran an instance of the Adcast software (i.e., a node). We used the topology manager to setup the network topology. Physical close nodes were connected together through simulated ad hoc physically links (here also called “physical links”) as it would have been the case in MANET environments. The messages coming from nodes not set as physical neighbors and not having physical links with the receiver are discarded by the topology manager. The network topology used (i.e., the position of the nodes in the physical space) is shown in Figure 4.7. The average number of physical neighbors is 3.2 per node.

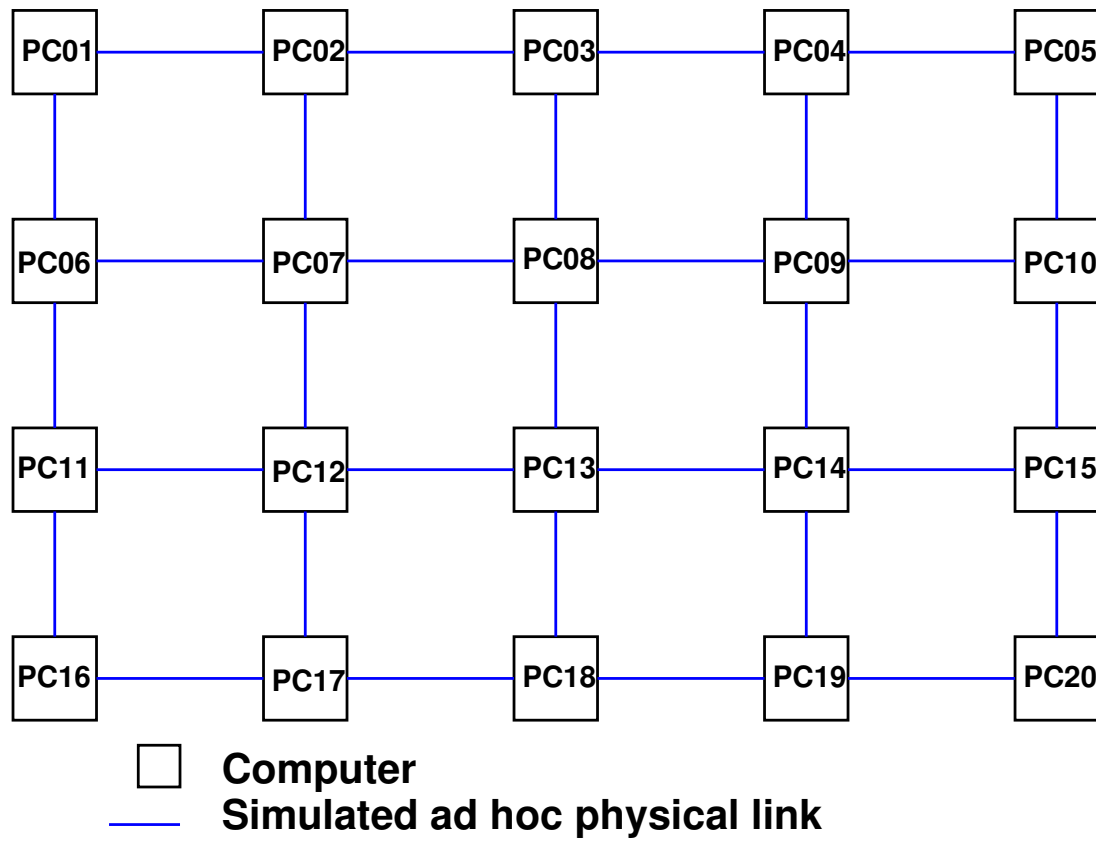


Figure 4.7: Experimental network's simulated topology

We started the DHT connection process from a centrally located node (8 or 13). So, the logical space will be spread among the neighbors by the connection algorithm. When all of the nodes were connected, we started a random request generator that produces 4 to 20 requests per minute. This represents a fast human usage. The results given by each node are calculated from the 30 last values obtained. In this way, we consider a warm-up phase to populate the routing table as done in the simulation environment. The experiment ran during 15 minutes.

For the simulation environment (for more details, please refer to Section 2.8.1), we used the following parameters:

- *Network size* : 20 nodes;
- *Connectivity* : the average number of physical connections of a node (network density) varies between 2 and 5 (in average 4);
- *Lookup requests* : for each experiment, the paths of 600 randomly generated requests are statistically evaluated;
- *Steadystate* : The simulation runs a warm-up phase during 2000 (2K) requests in average in order to reach a steady state before the statistical information is collected for the analysis;
- *Mobility* : Only static scenarios

In these experiments, we measured the cost in terms of in physical steps of a request, the stretch and the cache average size. The two first measurements give an overview of the lookup cost and efficiency. The last outlines the number of shortcuts gathered.

To evaluate the multicast tree building algorithm, we started the system and ran the DHT connection process as for the DHT experiments, but without the random request generator. Then we started the multicast layer on all

devices. When all the devices ran all the layers, we connected 8 nodes one after another to a test group. We extract the tree structure 5 minutes after the last observed modification to have a steady tree structure.

For the multicast tree building algorithm, we evaluated the effect of the rearrangement methods on the tree structure. This gives indication on whether if the algorithm is able to adapt the tree structure to the network topology as expected, and whether close members get connected together. We do not present numerical results because the multicast tree is too small to obtain valuable results.

Presented results were obtained from the experiments of our algorithms with all proposed improvements (see Chapters 2 and 3).

4.4.2 Experimental results

Distributed hashtable

The results obtained with Adcast and the simulation are presented together in Table 4.1. The Δ is the difference between the reference implementation (here the simulation) and the Adcast software in percent. It is interesting to note that the DHT supported by Adcast behaves like the DHT implemented for simulations. The results obtained by simulations and the Adcast experiment are really close. The maximal difference between the obtained results is less than 8%. This confirms the behavior observed in larger simulations (i.e., the simulator used to evaluate our algorithms provides realistic results).

In particular, a single Δ is larger than 5%, the average number of physical steps per request. As shown in Table 4.1, the average number of physical steps required to achieve a request is bigger for the simulation environment than for Adcast. This is due to the connection algorithm used in Adcast. Indeed, in Adcast, a node joins the DHT by connecting to a physical neighbor. In this

	Simulation	Adcast	Δ
Avg # of phys. steps for requests	2.88	2.65	-7.98%
Avg # of phys. steps for answers	2.74	2.64	-3.64%
Stretch (lookup vs direct route)	1.05	1.003	-4.47%
Avg cache size	18.09	18.75	3.52%

Table 4.1: Average use of logical shortcuts

manner, one of the logical neighbors of the connecting node is also a physical neighbor. However, in the simulation environment, the nodes receive random DHT identifiers and are placed randomly. So the probability that a logical neighbor is also a physical neighbor is higher in Adcast than in the simulation environment. As a result, the requests' route may be slightly longer in the simulation environment. This, of course, has also an influence on the stretch.

The stretch between the average number of physical steps required to achieve a request and the average number of physical steps between two nodes (direct route) is close to one for both experiments. This is actually not surprising. Indeed, the results obtained in Section 2.8.2 show that the stretch value increases when the number of nodes increases. Consequently, the stretch value must be small in experiments implicating only a small number of nodes.

Multicast tree building algorithm

We present here two multicast trees built on the experimental network. For each tree, we show its state just after the connection of all the members (Figures 4.8 and 4.10) and the state at the end of the experimentation (Figures 4.9 and 4.11).

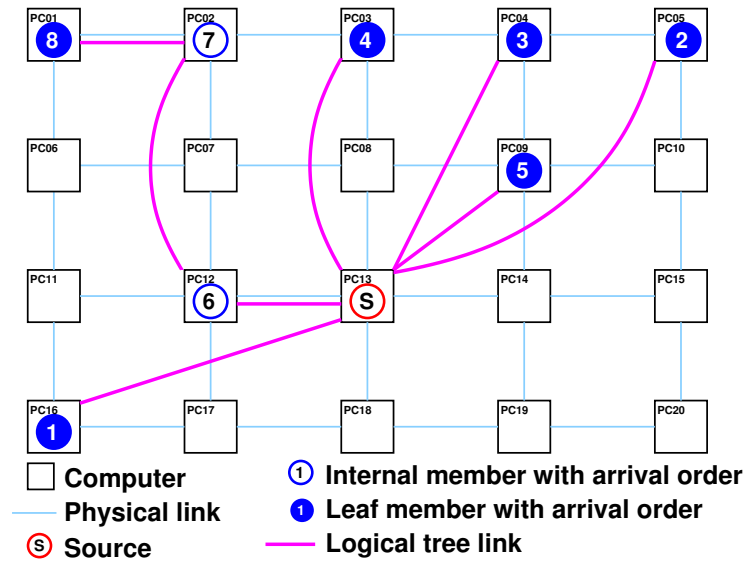


Figure 4.8: Experimental multicast tree #1 after connection

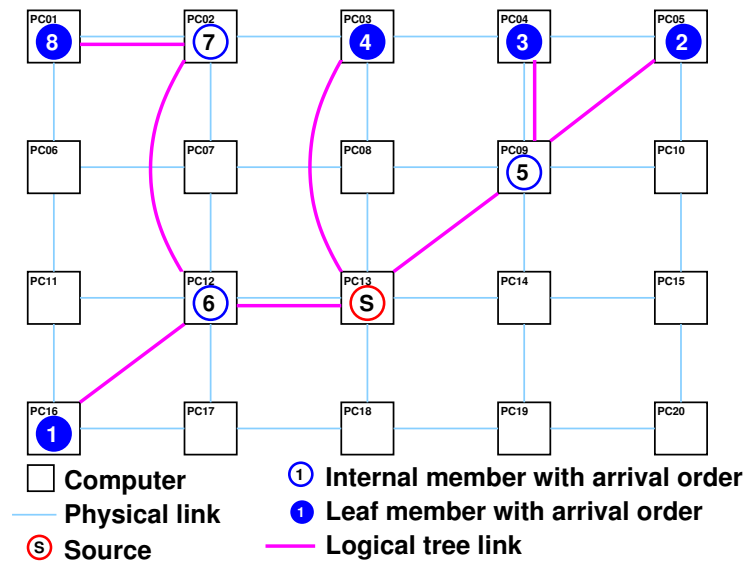


Figure 4.9: Experimental multicast tree #1 at the end of the experiment

In Figures 4.8 and 4.10 presenting the state just after the connection, we can see that the connection procedure works as expected. Indeed, in Figure 4.8 node 7 connects to node 5 that is already a member of the tree and node 8 connects to node 7 afterwards. This is also the case for the tree presented in Figure 4.10, node 6 connects to node 5 that is already a member of the tree. This confirms that the connection works as expected and a connecting node becomes the child of a tree members when possible.

Furthermore, in both trees, most of the members are directly linked with the source after the connection procedure. This behavior has already been observed in the multicast tree building algorithm simulations. Consequently, the simulations used to validate our algorithm have realistic results and Adcast behaves identically.

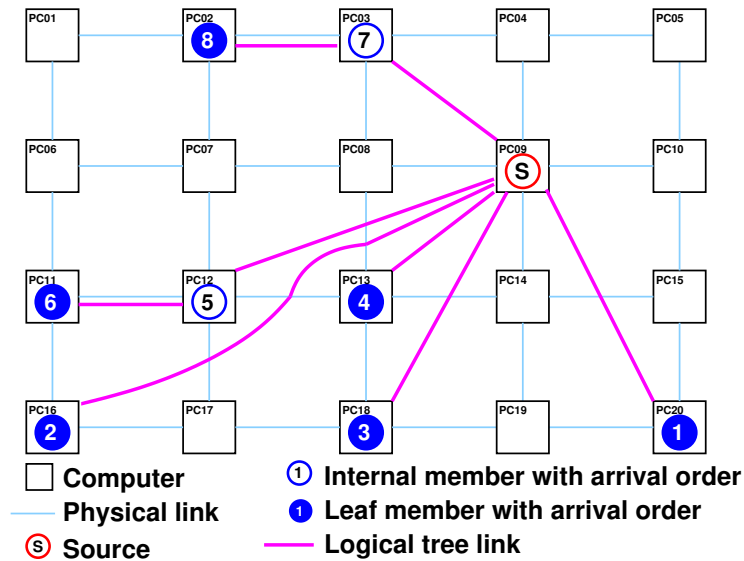


Figure 4.10: Experimental multicast tree #2 after connection

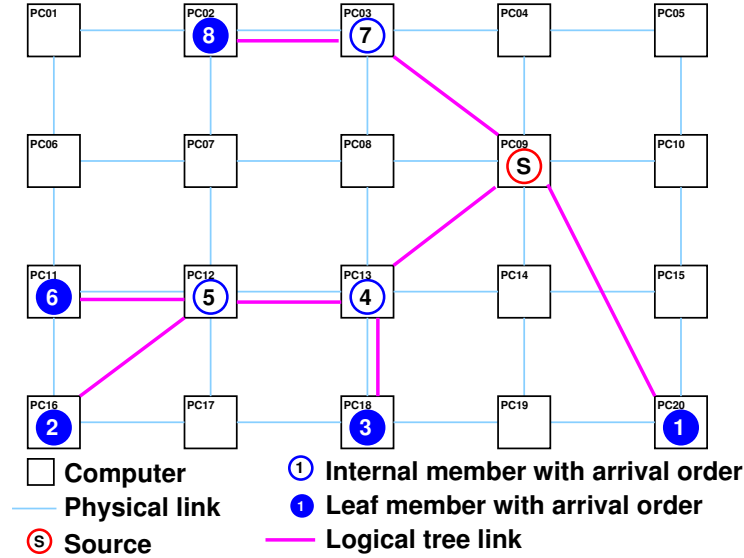


Figure 4.11: Experimental multicast tree #2 at the end of the experiment

At the end of the experiment, the structure of both trees (#1 and #2) has been improved by our algorithm. In the improved tree presented in Figure 4.9, member 5 became the parent of members 2 and 3. Consequently, close members get connected together, the load (membership management and multicast distribution) on the source S decreases and the membership management is spread among the participating members. In tree #2 presented in Figure 4.11 also, close members get connected together. Members 2 and 5 get connected to member 4, and member 2 becomes a child of member 5. After the arrangement process, the source has only three direct children (six after the connection process). Thus the source load is divided by two and the membership management is spread among the participating members. In both trees, the structure is modified as expected and the final tree structure fits the network's topology. This confirms that the improvement of our algorithm works in a real environment. Moreover, these behaviors have been observed in the simulation environment as well, validating the obtained results.

4.5 Summary and discussion

In this chapter we presented Adcast, the software that implements the algorithms detailed in Chapters 2 and 3 in a real MANET environment. Adcast is divided in independent layers providing and requesting features from each other and communicating through defined interfaces.

The DHT algorithm presented in Chapter 2 has been implemented in Adcast's lookup layer. It has been adapted to work in parallel and handles the requests as soon as they arrive. It is able to work either independently or to provide an efficient lookup service for other layers with optional features like interrupted routing. A connection algorithm has been developed and integrated in order to build the DHT overlay in the experimental environment.

Above the lookup layer, the multicast layer implements the multicast tree building algorithm presented in Chapter 3. The used implementation is an adaptation from the one used in the simulations. It works with sequential operations. It uses the lookup layer for its connection process and the routing layer for all other operations.

The Adcast software has been deployed on a set of 20 computers. The DHT and the multicast tree building algorithm have been evaluated in this environment. The results obtained for the DHT have been compared with the ones from a simulation parameterized to represent a real network.

The results are very close (i.e., less than 8% difference) and the expected behavior of the DHT lookup algorithm has been observed. This validates the simulation results. Moreover, two sample multicast trees have been built involving 8 members. The connection algorithm worked as expected and the obtained trees had the same characteristics than the one obtained in simulations. At the end of the experiment, as in simulations, both trees connect close members and distribute the membership management and the multicast distribution load among the members.

This implementation of the Adcast software worked as expected in the experimental environment. The experiment results encourage future developments of the software. Adcast contains innovative algorithms that have proven to be efficient for the ad hoc networks.

Finally, Adcast is a valuable source of observations for possible optimizations and future developments. The experimental environment brings new elements and challenges and consequently optimization and improvement opportunities.

Conclusion

Nowadays, the widespread use of ad-hoc networking is still limited, but this technology is steadily growing, supported by a great deal of research conducted to improve device and application efficiency. Their typical applications like data or information collection and aggregation often rely on flooding, allowing the development of quick and easy solutions. However, to develop solutions for information lookup and distribution, the flooding approach has major drawbacks, including the network load generated, high transmission costs and the number of nodes involved. Moreover, flooding algorithms do not scale well, and they are often prohibitive due to the number of messages generated and the time needed to transmit them. Finally, many communications and the corresponding messages handling consumes scarce resources like energy, leading to a global network lifetime reduction.

In this thesis, we proposed two novel algorithms that avoid flooding and are specifically adapted to MANETs: a distributed hash table (DHT) and a multicast tree building algorithm.

5.1 Contributions

The principal contribution of this thesis consists of two specialized algorithms for MANETs. They operate according to the P2P paradigm where each participant supports an instance of the algorithm. A logical overlay is built on top of the physical network. Our first algorithm, manages a logical space providing lookup facilities. Our second algorithm establishes a multicast tree to efficiently distribute content.

The algorithms and the experimental results are summarized in the two following sections.

5.1.1 Distributed hashtable for MANETs

Distributed hashtables have demonstrated their efficiency to locate existing content on the Internet. MANETs can take advantage of such a distributed lookup infrastructure to efficiently locate content. However, adapting well-known, Internet based approaches like Chord [Stoica 2001], Pastry [Rowstron 2001a] or CAN [Ratnasamy 2001] do not consider the nodes' positions when they create their logical overlay. Thus, the traffic required to maintain their routing tables will grow with the network size and may quickly overload the network (e.g., Chord has to maintain connections with $O(\log N)$ neighbors). Approaches with new concepts adapted for MANETs have been proposed, where the nodes are clustered or where the algorithm for building the routing table has been adapted to consider physical proximity (e.g., [Araujo 2005], [Cramer 2005], [Pucha 2004b]). Only few approaches try to exploit the MANETs particularity to improve lookup performances and reduce the cost of the routing tables' management (i.e., load)(e.g., [Cramer 2005], [Castro 2006]).

We proposed a DHT organizing nodes in a logical ring as done for instance in Chord [Stoica 2001]. To maintain the ring, a node keeps track only of one logical successor and predecessor. Although these logical neighbors may be physically distant, their restricted number generates limited management overhead.

Deterministic long-range links, like in Chord, that have a prohibitive maintenance cost in MANETs are replaced by shortcuts spontaneously discovered in the physical neighborhood of the nodes traversed by the requests. Because in MANETs the neighbors' visibility is limited to a node's communication range, we also considered the neighbors of the neighbors (NoN) to increase the number of logical shortcuts available. This extension increases the number of available long-range links (i.e., shortcuts in the logical space) with limited cost in terms of number of messages.

Additionally, we exploited MANET specificities such as multi-hop routing and broadcast communication to gather and cache information from traveling requests. This information is then also considered when searching a node with the smallest logical distance to the desired key. This extension is particularly interesting as it only requires a limited amount of memory.

After having demonstrated that our algorithm converges and terminates in all cases, we evaluated its performances by the means of simulations. The algorithm has been tested in static and mobile scenarios with different network sizes.

The obtained results demonstrate that the long-range links discovered thanks to MANET specificities can be often used along a requests path. Consequently, the requests are efficiently routed in the logical space and quickly converge to the node responsible for the desired key. The quick logical convergence limits the average number of physical steps needed by a lookup, hence also limiting the global resource usage.

Finally, we implemented the DHT algorithm in the Freemote emulation and the Adcast environments. The comparison of the results obtained in the three environments exhibited only small differences mostly due to implementation differences. This confirms the realism and the scalability of the simulation results.

5.1.2 Multicast tree building algorithm for MANETs

Besides flooding or controlled flooding approaches, information distribution in MANETs requires a membership management process and a distribution infrastructure. Solutions based on DHTs (e.g., [Passarella 2006], [Carvalho 2006]) use the lookup overlay either to locate and distribute the content, or to locate the node (i.e., group of nodes) responsible for the content distribution. They do not consider the sharing of the membership management and the message distribution load.

We proposed a multicast tree building algorithm working upon the DHT for MANETs. The algorithm uses the lookup service provided by the DHT to find a tree member (in all cases and at least the source) to connect to. A node interested in a multicast group looks up the DHT with the group name as key. Then at each node along its path, a request is analyzed. If the processing node is a tree member, a connection proposal is sent to the requester until the request reaches the node responsible for the key (i.e., the multicast source). The connection algorithm also takes advantage of the specificities of MANETs. Indeed, tree members listen to the network communications passing nearby and propose themselves as parents if a connection request is intercepted.

Besides the optimized connection algorithm, we developed several methods to improve and maintain the tree structure. These methods use the traveling multicast messages and tree heartbeats (overheard or passing through) to gather information on the local tree structure. This information is then used to

rearrange the tree structure in order to fit as well as possible to the underlying physical topology.

To evaluate our multicast tree building algorithm, we simulate static and mobile scenarios involving various number of nodes. The simulation results demonstrated that our algorithm builds and maintains multicast trees connecting physically close members together. Consequently, the number of relay nodes required to distribute the content gets significantly reduced. The simulation results have also shown that the rearranged tree spreads the membership management and the message distribution load among the members. Generally speaking, these results show that our multicast tree building method distributes the global load among the member nodes and limits the global multicasting cost compared to other approaches, e.g., the shortest path tree.

Finally, we carried out experiments on the algorithm with Adcast in a real setup. The algorithm worked as expected and build multicast trees fitting to the physical network topology. These results confirm that the simulation results are in line with real behavior.

5.2 Discussion

The algorithms have been designed to take advantage of the network specificities rather than being hampered by them. The simulation results demonstrated the efficiency of this approach for both algorithms. Both algorithms achieved good performance in the simulated MANETs with the help of local information.

In particular, logical shortcuts found in the close physical environment and gathered information enable the DHT algorithm to obtain sub-linear lookup performances with a reasonable routing stretch compared to direct path between the requesting and the requested node.

The multicast tree building algorithm benefits as well from the environments' specificities. The experiments have demonstrated that the message relaying and the broadcast communication can be exploited to deduce knowledge on the tree structure that can be used to build and maintain multicast trees connecting close members together, with a structure closely related to the network topology.

The algorithms have been designed to rely mostly on local information and to avoid unnecessary communications. Thereof, costly operations like network wide communication requiring heavy routing processes are most of the time avoided or really limited. This allows to keep the consumption of scarce resources at a minimum.

At the same time, the experimental results exhibit that the DHT algorithm running with all extensions significantly reduces the lookup path cost in terms of physical steps. Less transmissions implies less bandwidth, CPU and energy consumption. Similarly, the simulations have shown that the memory and distribution load are well spread among the participating nodes running the multicast tree building algorithm. Additionally, the multicast cost of the tree built with our algorithm is lower than the multicast cost of other approaches like shortest path tree. Consequently, the number of transmissions required to distribute the content is also lower, thus minimizing the global resource consumption. Generally speaking, both algorithms have therefore demonstrated their capacity to limit their requirements in memory, CPU, and energy.

Despite the good results obtained, the Adcast implementation has brought out the redundancy of certain operations among the different algorithms (for example each layer sends its own heartbeats). These operations are simple and small service routines that maintain the base structure necessary to support algorithms' operations. Independently taken, they represent a small and quite negligible load. But when running all together on a real MANET, as done for

example in Adcast, their effect is no longer insignificant. Besides the optimization of each algorithm for the MANET environment, the possible inter-layer synergies must be considered to build efficient applications for MANETs.

5.3 Future work

In this thesis, we studied and evaluated the general concepts allowing DHT and multicast tree building algorithms to operate efficiently in MANETs. Some algorithmic parts still must be developed to get a fully operational and global system. In particular for the DHT, algorithmic solutions must be explored for reliable and efficient arrival and departure procedures.

The MANETs are unstable networks, suffering from frequent unexpected disconnections (churn). Besides normal arrivals and departures, algorithms operating in this specific environment must be designed to support frequent and unexpected nodes' disconnections. They have to be able to maintain efficient operations in such unfriendly settings. The DHT algorithm then requires mechanisms to support churn. As for the lookup algorithm, these mechanisms have to be developed specifically for MANETs, in order to be able to efficiently operate in this particular environment.

The multicast tree building algorithm also requires mechanisms to support frequent and concurrent changes in the membership. Adaptations should be found to allow these membership changes to be handled locally when it is possible, and to avoid a large number of expensive communications when it is not possible. These mechanisms have to be designed to support higher mobility, thus increasing the application scope of the algorithm.

The presented lookup algorithm demonstrates the efficiency of finding logical shortcuts in the nodes' physical neighborhood. The neighbors and cache entries selection method remains trivial in our approach. Optimizing the kept

entries is still an open and interesting question that can be easily coupled with churn resistance. Preliminary work for improved caching has already been done and can be found in [Castro 2009]. Finally, analytical studies are needed to confirm the observed performances and to determine the limitations of these approaches.

In this thesis, we demonstrated the efficiency of our algorithm in building and maintaining a multicast tree. In reality, the use of such algorithms in everyday application requires many multicast trees to be built and maintained at the same time. A trivial approach will be to simply build them as with the existing algorithm. This is actually not a suitable solution in MANETs because although optimized and efficient, the super-imposed multicast trees (one per group!) have common and redundant parts and generate high network load, thus consuming a lot of resources. Consequently, solutions optimizing and merging the common and redundant parts of multicast trees built on the same network should be explored. Solutions to these kinds of problems are mandatory in order to make this system truly scalable and efficient to support wide spread applications in real MANET environments. Although we consider that churn handling is supported by the lookup protocol, it also may be an open issue. Indeed, the multicast tree building algorithm could be improved to locally reconnect broken branches without starting a full connection procedure.

The lessons learned with the Adcast implementation are very interesting. It has been shown that simple service messages necessary in real implementations can also be used to freely gather information for further structure improvements. Moreover, the implementation of our algorithms in the different layers of the Adcast software brought out the real cost of redundant and small service processes. These processes are common to all of our algorithms. Similar processes are probably also used in many other approaches designed for MANETs. Thus, studying their real cost and optimization possibilities remains a challenge, that is often overlooked. In real environments, the delay and latency

are also challenges that must be investigated in large deployments of MANET applications. The solutions grouping the services presented here as independent (routing, lookup, message distribution) represents an open field. Although already explored, the design of the algorithmic solution using the ideas presented in our algorithm has not been realized until now to the best of our knowledge.

MANETs are suitable for environments where no central node or network infrastructure can be used to support the networking operations. In particular, they are well adapted for environmental monitoring (in buildings or outside) as they can be quickly deployed with minimal configuration. Typical devices are equipped with various sensors (light, temperature, ...) and provide their measurements through multi-hop paths to a node collecting them. But they can also be used for collecting data in real time on mechanical systems where a classical wired approach will be unsuitable (e.g., the remote monitoring of an old boat engine without making holes everywhere for the wires or monitoring of engines' moving elements). Our algorithms may be implemented and adapted to support specific information distribution among nodes to produce adaptive behaviors (of system parts) or to quickly and easily contact specific nodes for automatic actions. MANETs are also used for emergency situations after disasters or military conflicts. Our DHT algorithm may be improved to support information storage and lookup, thus allowing people to get news from relatives, to locate appropriate medical centers, etc. The multicast tree building may be implemented as a good solution to distribute relevant information to the adequate team in such environments. Indeed, giving the correct information quickly may improve rescue operations. Finally, many other applications of such algorithms should emerge with the arrival of powerful and highly mobile devices like the tablet PCs.

APPENDIX A

Freemote

This great work has been realized by Timothée Maret for his Master Thesis [[Maret 2007](#), [Maret 2008](#)].

Timothée Maret TIC Institute, University of Applied Sciences of Fribourg
Switzerland timothee.maret@epfl.ch

A.1 Introduction

P2P systems first emerged as overlays stacked on top of the Internet for many useful applications including file sharing, VoIP, distributed data storage (Freenet), collaborative tools, etc. [[Khan 2008](#)]. Their success is usually based on the peer-to-peer property intrinsic to the Internet, providing transmission routes between end-nodes. In MANETs, the situation is different because end-nodes can also act as routers; or more precisely wireless routers. This means that each end-node/router has at least one wireless link to all physical neighbors defined by its radio coverage area. MANETs also differ from the classical Internet because their wireless links are usually much less reliable than fixed links, and are often impaired by congestion. Also, assumptions generally made regarding bidirectional links may not be true, given differences between nodes caused by implementations, remaining battery power levels, time variances, physical environments, etc. Finally each end-node/router's participation also varies in time in certain cases, due to its joining or leaving the MANET for a variety of reasons: changes in geographical position, turning itself off for some period of time to reduce power consumption, etc. The MANET node's wireless router role can

however be seen as an advantage, and for this reason many innovative algorithms have been designed for them. One possible goal for these algorithms is to ensure smooth scaling when applied to very large number of nodes and transmissions. Simulations must often be used however to evaluate P2P/MANETs applications, given the difficulty and expense involved in deploying a real, large-scale testing network. Also, monitoring of nodes in a real network is a time-consuming and complex task, given that nodes are usually distributed geographically, communicate over changing routes and might only provide few output modalities (often limited to a few LEDs) used to track execution, even during testing phases. Moreover, the deployment an ad-hoc network in real size for testing purposes requires extensive resources: equipment, space, time and manpower. Thus, since this phase cannot usually be actually conducted it may in some cases be replaced by specially designed software tools and frameworks able to run repeatable experiments in controlled environments or provide meaningful numerical values based on averages and statistical analyses. Due to time and resource constraints however, even experiments such as these cannot be always conducted.

Once a P2P algorithm has been designed and its performance demonstrated analytically or through simulation, it would be ideal to take the proposed algorithm's source code and simply port it and run in an environment closer to reality, but without the irreproducible effects encountered in real environments. The same source code would have to be used to avoid any potential differences between the specifications and implementation.

With the above-mentioned facts in mind, we developed the Freemote emulator in order to provide a suitable platform on which to test P2P applications and algorithms designed for MANETs of any size (a few or several thousands nodes). To ensure their practical suitability and behavior credibility in the simulated environment, we decided to link the emulated nodes with real nodes, thus forming an integrated ad-hoc network environment. These capabilities were thought to be the most important ones compared to other last-step performance

evaluations preceding full-scale deployment in a real environment. Thus, to facilitate many aspects related to software development we chose the Java language. This choice is also justified given the availability of Java programmable JMote prototypes developed at the University of Applied Sciences in Fribourg (www.eia-fr.ch). This Freemote emulator is able to run the same Java source code on the real and emulated nodes, thus bringing experimental results closer to reality.

Given that Java programmed nodes are not widely used, we chose a transmission format compatible with the popular Berkeley motes programmed using the nesC programming language [Nes 2008]. This inter-nodes communication is in fact based on the TinyOS [Tin 2008] message format, allowing the well-known MICAZ and TelosB Berkeley type nodes to be used as real nodes.

This mixing of emulated and real nodes was carried out via a "bridge" node attached to the workstation running the emulation. The emulated and real nodes communicate over this dedicated bridge, which operates as a normal node during the experiment. The real nodes could be any nodes compatible with the bridge node. At the physical level, the current Freemote system uses hardware based on any IEEE 802.15.4 compliant radio chip, and can thus handle both our Java nodes (JMote) and the well-known Berkeley motes.

To ensure Freemote emulation system's scalability, emulated nodes can be distributed on many workstations connected in a LAN in a classic configuration (i.e. TCP/IP). Experiments have shown that the Freemote system is able to support emulated systems having up to 10,000 nodes, distributed over a number of networked computers. The maximum number of nodes is currently defined by the addressing scheme chosen, set at 65,536 addresses. The Freemote software also includes a Graphical User Interface (GUI), thus making it possible to set up the desired experiment in an intuitive manner and easily follow the emulation using the information provided.

The Freemote system is readily available on the Internet [fre 2009] and can be tuned or modified to allow experiment with any P2P algorithms or applications designed for networks having "small" nodes or motes.

This chapter on our Freemote testbed is organized as follows: Section A.2 reviews background information about MANET simulators and emulators, as well as their suitability for P2P applications. Section A.3 gives a detailed description of the Freemote emulator system. Finally, the chapter is summarized in Section A.4.

A.2 P2P mobile ad-hoc system testbeds

This section presents an overview of the various approaches related to MANET simulations and emulations, and also discusses their suitability for P2P applications.

Although numerous tools are available for analyzing P2P and MANET applications, none of them is suitable for all the experiments being carried out by researchers. We thus have to select the tool that will provide the most realistic results for a specific setup. As pointed out in [Göktürk 2007] these tools can be classified into three categories, according to their level of realism, which varies from less accurate simulators to emulators and finally to highly accurate testbeds.

Current tools available for evaluating MANET algorithms include real Berkeley mote networks such as Emulab [Johnson 2006] and Motelab [Werner-Allen 2005]. One instance of these networks can be shared between many research teams, with each one remotely controlling and using the network during a given number of hours or days. These tools provide useful features that help reduce the time needed for experimental setups and allow scenarios to be reproduced during

experiments, given that a node's motion and initial position can be configured. P2P experiments carried out with the help of these tools resemble real situations quite closely but only involve a relatively small number of nodes.

ATEMU [Polley 2004], Avrora [Titzer 2005] and MSPsim [Eriksson 2007] are "fine-grain simulators" which operate at the instruction level through simulating Berkeley mote processor instruction sets such as the MSP430 or AVR chip family. Experiments with these simulators can handle several hardware setup variations, including aspects related to timing or power consumption, and can also provide highly accurate evaluation results. However, they require high computational power and provide only basic visualization tools. Thus "fine-grain simulators" are usually too detailed to investigate P2P applications over a large MANET. TOSSIM [Levis 2003] is a step-by-step discrete event simulator for TinyOS applications which when coupled with TinyViz makes for an extensible visualization tool. It runs an unmodified NesC code application and simulates the TinyOS behavior of those components tied to hardware. Although TOSSIM has some of the same drawbacks mentioned above, it is useful for testing the basic functionalities of node software for a given P2P application, but not for investigating overall behavior across a complete network.

The EMStar [Girod 2007], SENS [Sundresh 2004] and COOJA [Osterlind 2006] more closely resemble the Freemote emulator environment previous ones, as they provide a less accurate simulation model for the low level and radio parts. Moreover, they focus more on network behavior analyses than on time-based performance evaluations. SENS is similar to the Freemote emulator, providing a layered and modular environment able to run applications (composed of interchangeable modules) written in C++. Also, the code written for the emulated nodes is not directly executable on real nodes but must be ported to Berkeley motes. Not only does SENS use a different programming language (C++ instead of Java), it differs from the Freemote emulator in that it does not support any bridges to real nodes.

COOJA [Osterlind 2006] is a Java-based simulator for the popular Contiki OS [Dunkels 2008] and is able to mix simulated nodes at varying levels of detail within the same experiment. For prototyping it is able to simulate Java code, yet it is not executable on real nodes and thus must be adapted or ported. Furthermore, this simulator does not provide any bridges to real nodes, and thus the testing of P2P applications across MANETs is limited to simulated environments.

Finally the EMStar [Girod 2007] runs on Linux micro servers, providing an environment in which wireless network applications can be tested. EMStar provides the EmTOS facility enabling the execution of TinyOS applications written in NesC. The EMStar environment is highly versatile as it can mix microservers and emulated motes in the same experiment and can provide either simulated or real radio channels for connecting emulated nodes based on Berkeley mote arrays.

A.3 Wireless ad-hoc network emulation environment

To simulate or emulate their P2P application developments using MANETs, researchers often run a set of experiments, each associated with a predefined scenario and a specific environment configuration. The scenario defines the role of each network node as well as the application being tested. The environment configuration defines the network topology and node motion. Ideally scenarios and environments are identical and reproducible from one experiment, thus allowing testing and solution comparisons against the same conditions. The type of information gathered during an experiment depends on the application being tested. Generally speaking the information required includes (1) number of messages exchanged, (2) length of routes linking distant nodes, (3) network mean throughput and delay, (4) node power consumption and (5) code execution time.

"Fine-grain simulators" are not suitable for assessing applications on large MANETs. They need large amounts of computational power to maintain precise execution times and to provide non-relevant details such as power and memory consumption. For large network experiments high-level simulators should be considered instead. They limit the computational power required and still provide accurate information. Simulations may however have limited credibility if the effects of implementation cannot be taken into account. Emulations on the other hand offer greater control over implementation issues but are usually limited to one or a few nodes. The Freemote emulator was thus developed to combine the advantages of simulation and emulation, mixing not only these two aspects but also allowing real nodes to participate in the simulated network of emulated nodes.

In experimental and industrial MANET deployments, while the C language or nesC may have become popular, due to its hardware constraints, virtual machines (VMs) have emerged as new software environments for running applications on sensors [Barr 2004] [Levis 2002]. While VMs are less optimized to fit the limited hardware constraints of low-cost nodes than highly optimized operating systems (e.g., TinyOS) and languages (e.g., nesC), for developers they offer numerous advantages. In fact VMs such as Mate [Levis 2002] or SwissQM [Müller 2007] provide a simpler programming model, a smaller program footprint and well-defined programming interfaces, making development much simpler. These virtual machines run on top of optimized operating systems, provide extensible machine code and can support different programming languages. Although extensible, VMs have the disadvantage of providing a relatively small initial set of specialized programming functionalities. To make them more useful, programmers thus have to code special extensions for each specific application in the operating system's machine language. Highly optimized VMs have however been developed for well-known, portable and general-purpose programming languages such as Java (VM* framework [Koshy 2005], Squawk [Shaylor 2003],

Sentilla Point [Sentilla Corporation 2008]). The Freemote emulator is thus a suitable emulation tool for the emerging Java-based motes, while also running the same Java byte code on both emulated nodes and Sun's CLDC 1.0 compliant Java motes. Currently the code runs on JMotes, a CLDC prototype based on Java motes developed at the University of Applied Sciences in Fribourg (www.eia-fr.ch)[]. Although the Java code does not run on widely popular Berkeley's motes (e.g., TelosB, MICAz), the latter can also be used as real nodes or even to bridge nodes since the message transmission format chosen is based on TinyOS messages. Thus the experiments are able to run in a highly heterogeneous set of real nodes.

A.3.1 Freemote emulator

Figure A.1 shows the typical deployment of an experiment conducted with the Freemote emulator. Many nodes are emulated across various Java processes distributed over different computers and linked together on a standard LAN. In each process the number of nodes emulated is only limited by the computer's available memory and addressing space, currently set to handle up to 65,536 nodes. The emulated node topology is computed by the Topology Manager server and spread to all Topology Manager clients. This client-server approach ensures experimental scalability since it allows an unlimited number of clients. These clients can be run on very powerful workstations, thus guaranteeing simulation results within a reasonable time. Each workstation contains at least a Topology Manager client but only one machine will run the GUI and the Topology Manager server.

The Freemote emulator simulates the MAC and physical radio layers in a lightweight manner, ensuring reasonable running times. The focus on behavior credibility is achieved by mixing emulated and real nodes by means of a dedicated bridge. At the physical level, compatibility is ensured using the IEEE 802.15.4 radio standard, as in several Berkeley mote implementations.

Many P2P applications take it for granted that a physical node is accessible from any other node, as in an IP based network. With MANETs however routing remains a challenging research topic. The NST-AODV [Gomez 2006] mote implementation was used for testing purposes. A Java implementation of NST-AODV developed for the Freemote system was "optimized" for the IEEE 802.15.4 standard, because it uses MAC-level acknowledgements to detect broken routes. Our NST-AODV also implements an enhancement, enabling it to handle asymmetric radio links between nodes, as mentioned in the next section.

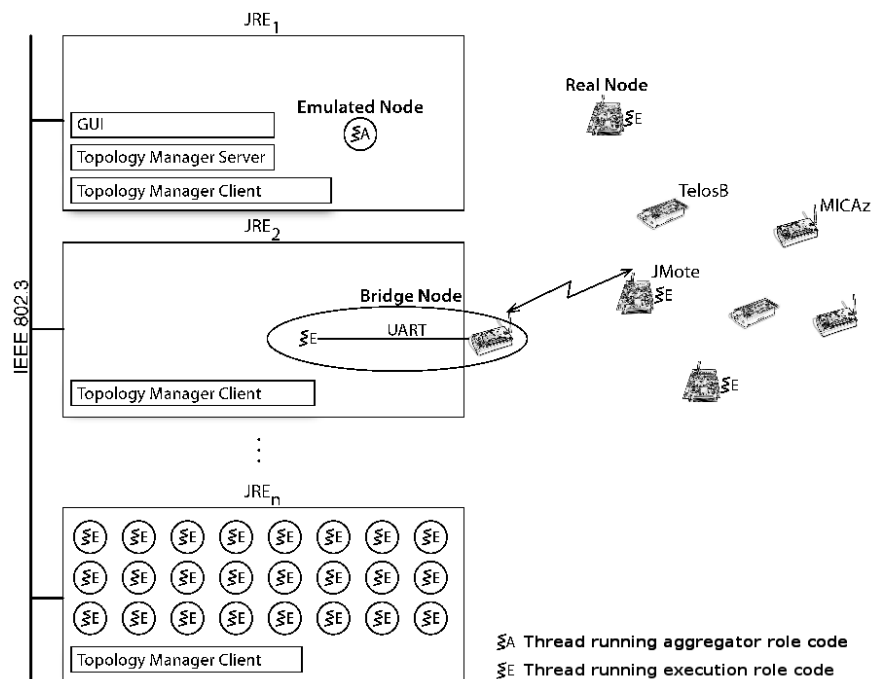


Figure A.1: Typical deployment during an Experiment

The Freemote emulator provides a powerful Graphical User Interface (GUI). As shown in Figure A.2, it displays the emulated network during runtime and the emulated nodes are shown in a square map, showing not only the node positions and their motions but also the physical links between nodes (delimited by

simple circles) and the content of the messages sent (lower part of Figure A.2). The GUI also displays and records all network operations. They are output in log format and thus are useful during the development for debugging complex network issues. Execution can also be slowed down, thus making it easier to visualize network behavior (the exchange of messages for example).

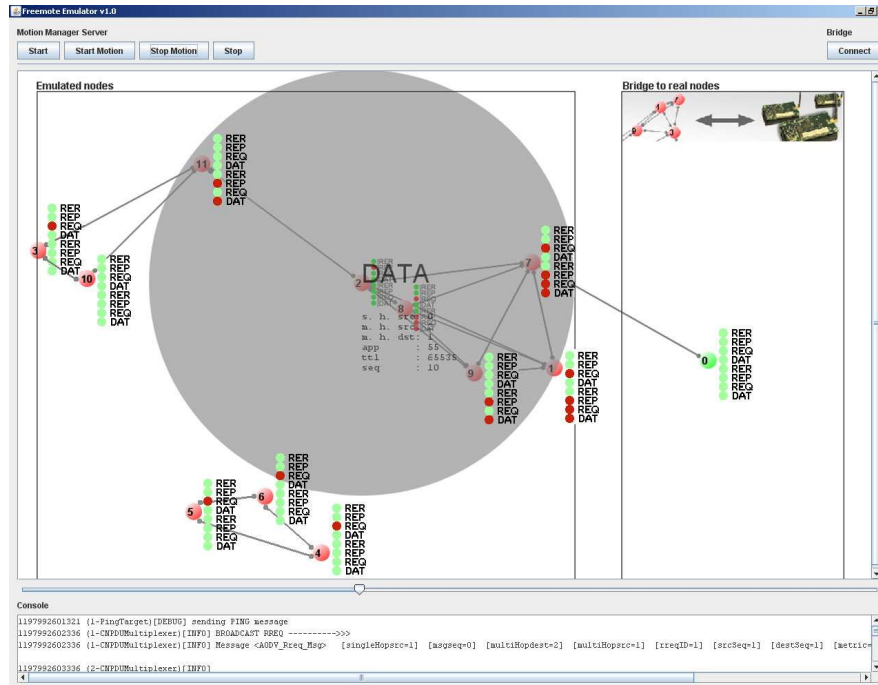


Figure A.2: GUI with emulated node visualization tool

As an example, Figure A.2 shows the information displayed during an experiment. The bridge node is represented in the right part and the emulated nodes are shown on the left part. The gray circle represents the radio range of the node with the address 2, which is sending a DATA message. The 8 LEDs of the real JMotes are also displayed for each emulated node in the Freemote GUI. It is thus possible to track behavior for both the real and emulated nodes.

A.3.2 Software implementation

The Freemote emulator software was designed for easy adaptation to P2P application investigations on MANETs and sensors networks. Figure A.3 shows the layered architecture design along with 3 layers representing the three node types (emulated, bridge and real nodes) that might take part in an experiment. These three layers are:

1. Usual physical and data link layers merged into the lowest layer. The IEEE 802.15.4 is used for real nodes and bridge nodes on the real network side. The UDP/IP/802.3 (Ethernet) is used on the LAN side to link the emulated nodes. TCP/IP is used to control the simulation parameters (e.g., position, radio coverage, transmission error rate) of the emulated nodes.
2. Routing layer, which currently implements the above-mentioned NST-AODV.
3. Classical transport and upper layers that are merged into the Freemote application layer. P2P research or MANET applications usually do not focus on end-to-end transport layers.

As usual the layered approach allows new sets of experiments to be built by mostly reusing code already developed. A cross-layer design is still possible since the developer retains full control of the interface between layers (Mux/Demux in Figure A.3). The implementations selected for each layer along with experimental parameters are defined in a single XML file. This file also contains sections on each node type (emulated, bridge and real) and defines a custom role for each network node. Typical experiments consist of two roles: aggregation and execution. Usually only one node carries out the aggregation role, intended to gather all experimental results. Many nodes are involved in the execution role, intended to run a specific P2P application and generate experimental results. The configuration file is easily created, stored and edited thanks to a GUI that maps all XML tags for graphic objects.

The multiplexers/demultiplexers define the interfaces between the layers as well as message sending and receiving methods. Each message sent on the data link and physical layer or routing layer is associated with an 8-bit label (Figure A.5). Typically used upon reception, this label differentiates one type of message from another one. On the lowest layer this information is directly mapped to the TinyOS active message type field, while a new field is added for the routing layer. Any application receiving messages must be programmed to register as a listener for the corresponding label on the multiplexer/demultiplexer. The lowest layer must then implement a promiscuous mode, allowing it to receive every message sent by physical neighbors, even if the message’s destination is not the current node.

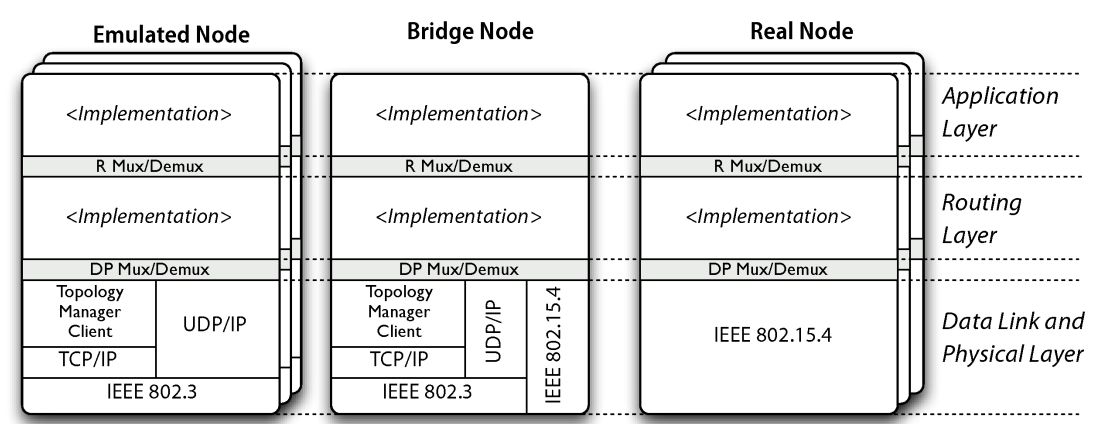


Figure A.3: Layered view of software architecture

The current Freemote software provides some sample applications such as the Ping-like test illustrated in Figure A.4, Figure A.5 and Figure A.6. These figures are not further detailed here but are shown to trigger the interest of P2P developers to implement their algorithms in their own Freemote testbed.

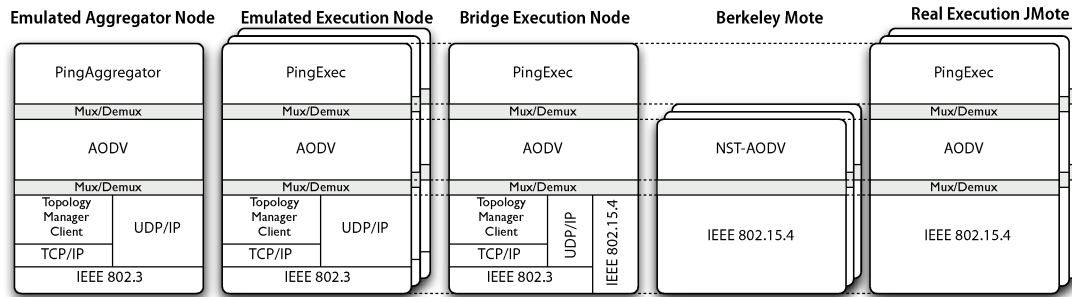


Figure A.4: Layered view of software architecture for “ping” application illustrated in Figure A.6

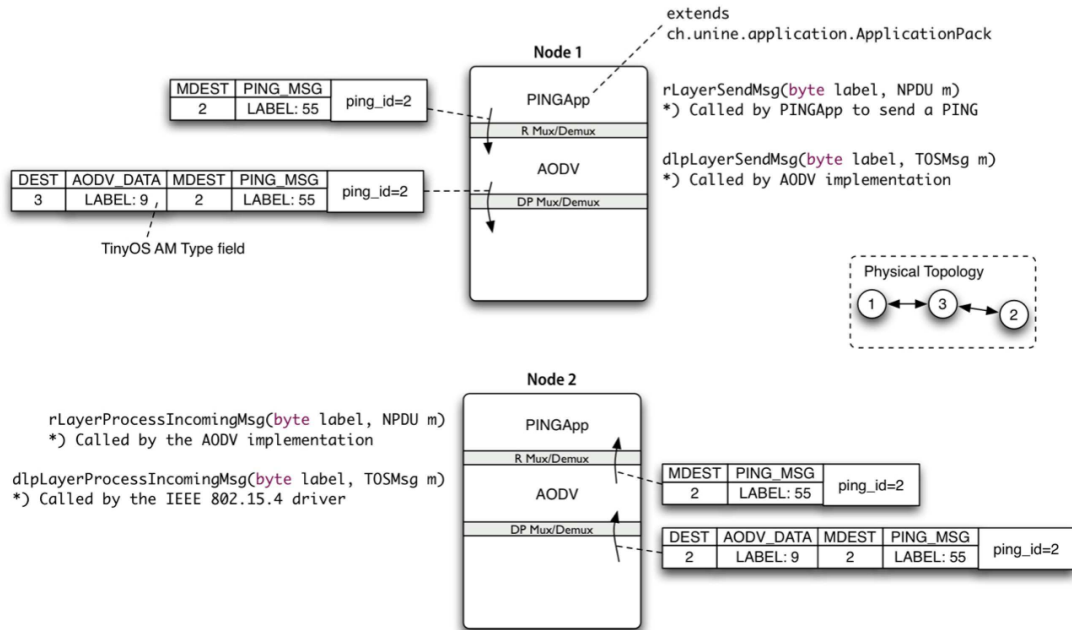


Figure A.5: Code templates used to facilitate implementation of each layer

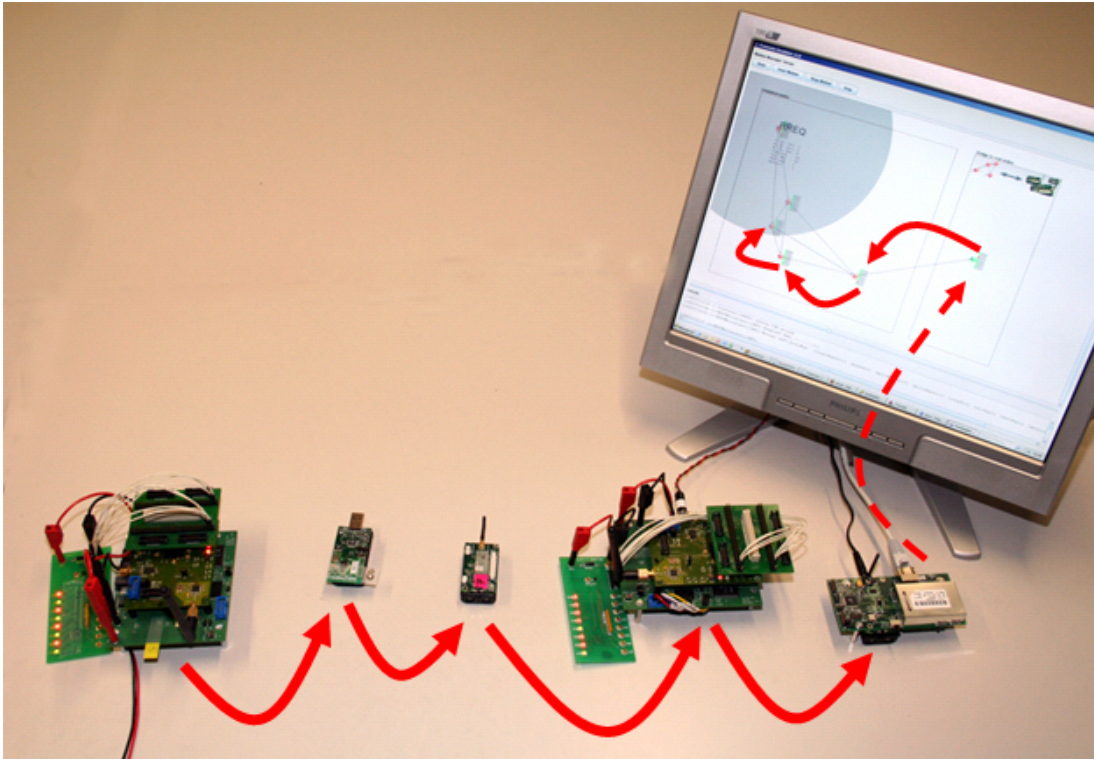


Figure A.6: Freemote experimental “ping” configuration mixing real and emulated nodes

A.3.3 MANET simulations

The emulated nodes and bridge node simulate IEEE 802.15.4 behavior in a lightweight manner, as characterized by the features summarized in Table A.1 below.

Feature	Definition
Transmissions errors	A random error rate is applied to each message sent, i.e., for each message generated by an emulated node. The node computes a random number to decide whether or not the message is effectively sent to all the neighbors. Future releases might improve this feature in order to compute a transmission error per neighbor, depending on the node-to-neighbor environment. An indoor coverage prediction tool developed might be used for this purpose [Echenard 2006].
ACK or Acknowledgments	IEEE 802.15.4 acknowledgments are not implemented but their effects are taken into account since a node verifies the radio coverage of each of its listed neighbors.
Promiscuous mode and broadcast	A sending node's neighbors receive a copy of each message sent (if no transmission error).

The bridge node interconnects the two data link and physical layers: one identical to the emulated nodes on the emulated network side and another one identical to the real nodes with which it must interface. On the emulated network side, the software interface uses the existing TinyOS Java API for sending and receiving messages over the serial port to a real mote mounted on a programmer board.

Feature	Definition
Radio range	Currently circular radio range for each node is fixed for the experiment's duration. Future releases might include radio range variations, according to the node's remaining power, other power limitations or geographical positions.
Unidirectional links	Physical links between two nodes could be unidirectional if one of the nodes has a radio range greater than the other.
Output power	The output power affects the radio range radius and can be different at each node. As discussed above, this feature is already included.

Table A.1: Data link and physical layer features

To simulate the radio medium's broadcast characteristic within its radio coverage, a node must know its physical neighborhood before sending a message, i.e., other nodes within its radio coverage. This list is provided to each node by the Freemote Topology Manager set up in a client/server manner. Each Topology Manager client (Figure A.1) stores topological information on the emulated nodes in the associated process. A single Topology Manager server maintains a Cartesian map containing all emulated nodes and periodically updates node positions according to the mobility scheme chosen. Currently, the following three mobility schemes are provided:

1. Fixed Model: Useful for repeatable experiments, able to define each node's specific angle, speed, initial position and radio range.
2. Random Uniform: Each node chooses a random speed within the [MIN-SPEED, MAXSPEED] range, along with an angle and initial position. All nodes have the same radio range.
3. Random Walk: Each node chooses a random speed and angle and then runs for a random time before choosing another set of parameters.

Once the node positions are computed, the Topology Manager server computes the new physical topology. The Topology Manager's algorithm first sorts the nodes along the X-axis and then in a single run through the list of nodes, creates all the physical bindings. It operates within $O [N (\log N + C)]$ time complexity, where C is the network's average connectivity and N is the number of emulated nodes. The bindings are unidirectional, as each emulated node can have its own circular radio range. The asymmetry in the radio links is used to simulate the acknowledgements (ACK) as explained below.

The Topology Manager server pro-actively pushes the topological modifications to the Topology Manager clients using TCP sockets to add or remove neighbors. This process guarantees that the Topology Manager clients always contain up-to-date topological information. Each node fetches a list of its physical neighbors from the Topology Manager client and then sends a copy of the message to each of its neighbors using UDP sockets. The sending node also checks to see if the message destination is within its neighborhood and if an acknowledgement is requested. It then verifies that the destination node is able to send it back (i.e., if there is a sending node in its neighborhood). This simulates the IEEE 802.15.4 standard efficiently by avoiding the sending of acknowledgement messages during the simulations.

The current IEEE 802.15.4 simulation comprises several limitations which might need to be improved if the following effects prove important: a) it does not simulate channel collision detection and thus the transmission error rate must account for those cases where two physical neighbors send messages simultaneously; b) it does not account for radio propagation delays, and considers neither the throughput nor path loss and fading in order to automatically adjust the transmission error rate.

A.4 Conclusion

The research on P2P and MANET include the use of simulations and also analytical studies. These two methods confirm expected performances and to reveal the benefits and limitations of the various innovative approaches. To ensure that any benefits are not lost during practical implementation, real and usually large testbeds have to be used, even though their cost and complexity are often prohibitive. Furthermore due to the lack of repeatability certain experimental results can be difficult to investigate. Emulation is therefore a more convenient way of assessing the behavior of innovative algorithms in real P2P application implementations on MANETs. The Freemote emulator was used to investigate improvements in P2P algorithms in realistic scenarios.

The Freemote emulator was designed to run the same code on emulated and real nodes, and to mix both nodes in the same experiment. It is thus hoped that the Freemote emulator could on the one hand provide an interesting solution for students and newcomers discovering MANETs and sensor network domains, allowing them to begin developing programs in the well know Java language. On the other hand for researchers and developers Freemote should prove an interesting tool because it allows the running of large-scale experiments. They might be distributed over a network of computers and as such provide remarkable execution performances. Furthermore, the possibility of mixing emulated nodes with real Java nodes and real Berkeley motes is expected to attract a certain amount of attention.

Bibliography

- [Araujo 2005] Filipe Araujo, Luis Rodrigues, Jorg Kaiser, Changling Liu and Carlos Mitidieri. CHR: A Distributed Hash Table for Wireless Ad Hoc Networks. In International Conference on Distributed Computing Systems Workshops, pages 407–413, Washington, DC, USA, 2005. IEEE Computer Society. [14](#), [47](#), [110](#)
- [Baccelli 2008] E. Baccelli and J. Schiller. Towards scalable MANETs. In ITS Telecommunications, 2008. ITST 2008. 8th International Conference on, October 2008. [16](#)
- [Barr 2004] Rimón Barr. An efficient, unifying approach to Simulation using virtual machines. PhD thesis, Cornell University, 2004. [125](#)
- [Borgia 2007] Eleonora Borgia and Franca Delmastro. Effects of unstable links on AODV performance in real testbeds. EURASIP Journal on Wireless Communications and Networking, vol. 2007, no. 1, pages 32–32, 2007. [37](#)
- [Bosneag 2005] Anne-Marie Bosneag and Monica Brockmeyer. GRACE: Enabling collaborations in wide-area distributed systems. In 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05), pages 72–77, 2005. [13](#)
- [Carvalho 2006] Nuno Carvalho, Filipe Araujo and Luis Rodrigues. Reducing Latency in Rendezvous-Based Publish-Subscribe Systems for Wireless Ad Hoc Networks. In ICDCSW '06, page 28, Washington, DC, USA, 2006. IEEE Computer Society. [47](#), [112](#)
- [Castro 2006] Miguel Castro, Matthew Caesar, Edmund B. Nightingale, Greg O'Shea and Antony Rowstron. Virtual ring routing: network routing inspired by DHTs. SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, pages 351–362, 2006. [14](#), [110](#)

- [Castro 2009] Marcel C. Castro, Andreas Kasser, Gabriel Kliot, Roy Friedman, Raphaël Kummer, Peter Kropf and Pascal Felber. Minimizing DHT Routing Stretch in MANETs. In *Adhoc 2009: 9th Scandinavian Workshop on Wireless Adhoc Networks*, Uppsala University, Sweden, May 2009. 116
- [Chen 2003] Xiao Chen and Jie Wu. Multicasting techniques in mobile ad hoc networks. CRC Press, Inc., Boca Raton, FL, USA, 2003. 46
- [Chen 2006] Bor-rong Chen, Kiran-Kumar Muniswamy-Reddy and Matt Welsh. Ad-hoc multicast routing on resource-limited sensor nodes. In *REAL-MAN'06: Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, pages 87–94, New York, NY, USA, 2006. ACM. 48
- [Cheng 2009] Hui Cheng, Jiannong Cao and Xiaopeng Fan. GMZRP: Geography-aided Multicast Zone Routing Protocol in Mobile Ad Hoc Networks. *Mob. Netw. Appl.*, vol. 14, no. 2, pages 165–177, 2009. 49
- [Clausen 2003] T. Clausen and P. Jacquet. RFC-3626 Optimized Link State Routing Protocol (OLSR), 2003. 16, 49
- [Conti 2004] Marco Conti, Enrico Gregori and Giovanni Turi. Towards Scalable P2P Computing for Mobile Ad Hoc Networks. In *PERCOMW'04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, page 109, Washington, DC, USA, 2004. IEEE Computer Society. 16
- [Cramer 2005] Curt Cramer and Thomas Fuhrmann. Proximity Neighbor Selection for a DHT in Wireless Multi-Hop Networks. In *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing*, pages 3–10, Konstanz, Germany, August 31–September 2 2005. 14, 110
- [Cramer 2006] Curt Cramer and Thomas Fuhrmann. Performance evaluation of chord in mobile ad hoc networks. In *MobiShare '06: Proceedings of the 1st*

- international workshop on Decentralized resource sharing in mobile computing and networking, pages 48–53, New York, NY, USA, 2006. ACM. 15
- [Das 2008] Saumitra M. Das, Himabindu Pucha and Y. Charlie Hu. Distributed Hashing for Scalable Multicast in Wireless Ad Hoc Networks. *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, pages 347–362, 2008. 48
- [Delmastro 2005] Franca Delmastro. From Pastry to CrossROAD: CROSS-Layer Ring Overlay for AD Hoc Networks. In *PERCOMW’05*, pages 60–64, Washington, DC, USA, 2005. IEEE Computer Society. 47
- [Devarapalli 2001] Vijay Devarapalli and Deepinder Sidhu. MZR: a multicast protocol for mobile ad hoc networks. In *IEEE International Conference on Communications*, volume 3, pages 886 – 891, 2001. 46
- [Dunkels 2008] Adam Dunkels. Contiki: A Dynamic Operating System for Memory-Constrained Networked Embedded Systems, 2008. 124
- [Echenard 2006] Nicolas Echenard and Jean-Frédéric Wagen. WLAN Service Coverage Based on PixelFlow Predictions, 2006. 133
- [eia] The Ecole d’ingénieurs et d’architecture de Frinourg website, Retrieved april 3, 2010, from <http://www.eia-fr.ch/>. 36
- [Eriksson 2007] Joakim Eriksson, Adam Dunkels, Niclas Finne, Fredrik Osterlind and Thiemo Voigt. Mspsim - an extensible simulator for msp430-equipped sensor boards. In *European Conference on Wireless Sensor Networks*, 2007. 123
- [fre 2009] The Freemote website, Retrieved July 27, 2009, from <http://www.assembla.com/wiki/show/freemote>, July 2009. 24, 122

- [Garcia Lopez 2009] Pedro Garcia Lopez, Raúl Gracia, Marc Espelt, Gerard Paris, Marcel Arrufat and Roc Messeguer. Topology-aware group communication middleware for MANETs. pages 1–10, 2009. [49](#)
- [Ge 2006] Min Ge, Srikanth V. Krishnamurthy and Michalis Faloutsos. Application versus network layer multicasting in ad hoc networks: the ALMA routing protocol. volume 4, pages 283–300, Amsterdam, The Netherlands, The Netherlands, March 2006. Elsevier Science Publishers B. V. [48](#)
- [Girod 2007] Lewis Girod, Nithya Ramanathan, Jeremy Elson, Thanos Stathopoulos, Martin Lukac and Deborah Estrin. Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks. Transactions on Sensor Networks, vol. 3, no. 3, 2007. [123](#), [124](#)
- [Gnu 2008] The Gnutella web site, 2008. [3](#)
- [Göktürk 2007] Erek Göktürk. A Stance on Emulation and Testbeds, and a Survey of Network Emulators and Testbeds. In European Conference on Modeling and Simulation, 2007. [122](#)
- [Gomez 2006] C. Gomez, P. Salvatella, O. Alonso and J. Paradells. Adapting AODV for IEEE 802.15.4 mesh sensor networks: theoretical discussion and performance evaluation in a real environment. In IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks, 2006. [37](#), [127](#)
- [Haas 1997] Zygmunt Haas. A new routing protocol for the reconfigurable wireless networks. In IEEE 6th International Conference on Universal Personal Communications Record, volume 2, pages 562–566, San Diego, CA, USA, 1997. [46](#)
- [Heer 2006] Tobias Heer, Stefan Gotz, Simon Rieche and Klaus Wehrle. Adapting Distributed Hash Tables for Mobile Ad Hoc Networks. In PERCOMW’06: Proceedings of the 4th annual IEEE international conference on Pervasive

- Computing and Communications Workshops, page 173, Washington, DC, USA, 2006. IEEE Computer Society. [15](#)
- [Jacquet 2005] Philippe Jacquet and Georgios Rodolakis. Multicast scaling properties in massively dense ad hoc networks. 11th International Conference on Parallel and Distributed Systems, vol. Volume 2, pages 93 – 97, July 2005. [66](#)
- [Jetcheva 2001] Jorjeta G. Jetcheva and David B. Johnson. Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks. pages 33–44, 2001. [47](#)
- [Johnson 1996] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In IEEE Mobile Computing, volume 353. Kluwer Academic, 1996. [13](#)
- [Johnson 2006] David Johnson, Tim Stack, Russ Fish, Daniel Montrallos Flickinger, Leigh Stoller and Robert Ricci and Jay Lepreau. Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. In Conference on Computer Communications (INFOCOM), pages 1–12, 2006. [122](#)
- [Karp 2000] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In MobiCom’00: Proceedings of the 6th annual international conference on Mobile computing and networking, pages 243–254, New York, NY, USA, 2000. ACM Press. [14](#)
- [Kaz 2008] The KaZaA web site, 2008. [3](#)
- [Kevin 2002] George Porter Kevin, Kevin Lai, Ion Stoica and Jeremy Condit. The Chord Ad-hoc Routing Protocol. Rapport technique, 2002. [12](#)
- [Khan 2008] I Javed Khan and Adam Wierzbicki. Foundation of Peer-to-Peer Computing. Elsevier Journal of Computer Communication, vol. Volume 31, no. Special Issue 2, 2008. [119](#)

- [Ko 2000] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. *Wirel. Netw.*, vol. 6, no. 4, pages 307–321, 2000. 37
- [Koshy 2005] Joel Koshy and Raju Pandey. Vm*: Synthesizing scalable runtime environments for sensor networks. In *Conference on Embedded Networked Sensor Systems*, 2005. 125
- [Koutsonikolas 2010] Dimitrios Koutsonikolas, Saumitra M. Das, Y. Charlie Hu and Ivan Stojmenovic. Hierarchical geographic multicast routing for wireless sensor networks. volume 16, pages 449–466, Hingham, MA, USA, 2010. Kluwer Academic Publishers. 48
- [Kummer 2006a] Raphaël Kummer, Pascal Felber and Peter Kropf. Distributed Lookup in Structured Peer-to-Peer Ad-Hoc Networks. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4276 of *Lecture Notes in Computer Science*, pages 1541–1554. Springer Berlin / Heidelberg, 2006. 6, 35, 37
- [Kummer 2006b] Raphaël Kummer, Pascal Felber and Peter Kropf. Practical DHT lookup in Mobile Ad-hoc Networks. In *Proceedings of the 4th MINEMA workshop*, Sintra, July 2006. 6
- [Kummer 2008] Raphaël Kummer, Pascal Felber and Peter Kropf. Building Multicast Trees in Ad-hoc Networks. In *MINEMA workshop, Autonomics 2008*, Turin, Italy, September 2008. 6
- [Levis 2002] Philip Levis and David Culler. Mate: a tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 85–95, 2002. 125
- [Levis 2003] Philip Levis, Nelson Lee, Matt Welsh and David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In

- Sensys: The ACM Conference on Embedded Networked Sensor Systems, pages 126–137, 2003. [123](#)
- [Maret 2005] Timothée Maret, Nicolas Martenet, Philippe Joye and Nicolas Schroeter. PDA Java: A Java Based Embedded System. Rapport technique, University of Applied Science of Fribourg, 2005. [8](#)
- [Maret 2007] Timothée Maret, Peter Kropf and Béat Hirsbrunner. Un environnement d’émulation hybride pour un algorithme de DHT adapté au contexte des réseaux ad-hoc. Master’s thesis, Universities of Fribourg and Neuchâtel, 2007. [8](#), [119](#)
- [Maret 2008] Timothée Maret, Raphaël Kummer, Peter Kropf and Jean-Frédéric Wangen. Freemote Emulator : A Lightweight and Visual Java Emulator for WSN. In 6th International Conference on Wired/Wireless Internet Communication, volume LNCS, pages 92–103, May 2008. [119](#)
- [Müller 2007] René Müller, Gustavo Alonso and Donald Kossmann. A Virtual Machine for Sensor Networks. In EuroSys, 2007. [125](#)
- [Nes 2008] The nesC website, 2008. [121](#)
- [Osterlind 2006] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne and Thiemo Voigt. Cross-Level Sensor Network Simulation with COOJA. In IEEE International Workshop on Practical Issues in Building Sensor Network Applications, pages 641–648, 2006. [123](#), [124](#)
- [Passarella 2006] Andrea Passarella, Franca Delmastro and Marco Conti. XScribe: a stateless, cross-layer approach to P2P multicast in multi-hop ad hoc networks. In MobiShare’06, pages 6–11, New York, NY, USA, 2006. ACM Press. [45](#), [47](#), [112](#)
- [Perkins 1999] Charles E. Perkins and Elizabeth M. Belding-Royer. Ad-hoc On-Demand Distance Vector Routing. In IEEE Workshop on Mobile Computing Systems and Applications, pages 90–100, New Orleans, Louisiana, USA, 1999. [8](#), [13](#), [37](#), [86](#), [90](#), [91](#), [92](#)

- [Polley 2004] Jonathan Polley, Dionysys Blazakis, Jonathan McGee, Dan Rusk and John S. Baras. ATEMU: a fine-grained sensor network simulator. In First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, pages 145–152, 2004. [123](#)
- [Pucha 2004a] Himabindu Pucha, Saumitra M. Das and Y. Charlie Hu. EKTA: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks. In 6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2004), pages 163–173, December 2004. [13](#)
- [Pucha 2004b] Himabindu Pucha, Saumitra M. Das and Y. Charlie Hu. How to Implement DHTs in Mobile Ad Hoc Networks? In the 10th ACM International Conference on Mobile Computing and Network (MobiCom), September 2004. [13](#), [110](#)
- [Ratnasamy 2001] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp and Scott Schenker. A scalable content-addressable network. In SIGCOMM’01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pages 161–172, New York, NY, USA, 2001. ACM Press. [3](#), [110](#)
- [Rowstron 2001a] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In Middleware 2001, volume 2218 of Lecture Notes in Computer Science, pages 329–350. Springer Berlin / Heidelberg, 2001. [3](#), [4](#), [11](#), [13](#), [16](#), [17](#), [42](#), [110](#)
- [Rowstron 2001b] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro and Peter Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In Networked Group Communication, pages 30–43, 2001. [4](#), [45](#), [53](#)
- [Royer 1999] Elizabeth M. Royer and Charles E. Perkins. Multicast operation of the ad-hoc on-demand distance vector routing protocol. In MobiCom ’99, pages 207–218, New York, NY, USA, 1999. ACM Press. [53](#)

- [Ruiz 2005] Pedro M. Ruiz and Antonio F. Gomez Skarmeta. Approximating optimal multicast trees in wireless multihop networks. 10th IEEE Symposium on Computers and Communications, 2005. ISCC 2005. Proceedings, vol. 27-30 June 2005, pages 686 – 691, 2005. [50](#)
- [Sanchez 2006] J.A. Sanchez, P.M. Ruiz and I Stojmenovic. GMR: Geographic Multicast Routing for Wireless Sensor Networks. Sensor and Ad Hoc Communications and Networks, 2006. SECON'06. 2006 3rd Annual IEEE Communications Society on, vol. 1, pages 20 – 29, 2006. [48](#)
- [Seet 2009] Boon-Chong Seet, editeur. Mobile peer-to-peer computing for next generation distributed environments: Advancing conceptual and algorithmic applications. Information Science Publishing, May 2009. [6](#)
- [Sentilla Corporation 2008] Sentilla Sentilla Corporation. Software Architecture, 2008. [126](#)
- [Shaylor 2003] Nik Shaylor, Douglas N. Simon and William R. Bush. A Java virtual machine architecture for very small devices. In Special Interest Group on Programming Languages, pages 34–41, 2003. [125](#)
- [Stoica 2001] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pages 149–160, New York, NY, USA, 2001. ACM Press. [3](#), [11](#), [17](#), [42](#), [110](#), [111](#)
- [Sundresh 2004] Sameer Sundresh, Kim Wooyoung and Gul Agha. SENS: A Sensor, Environment and Network Simulator. In Simulation Symposium, pages 221–228, 2004. [123](#)
- [Tin 2008] The TinyOS website, 2008. [121](#)

- [Titzer 2005] Ben L. Titzer, Daniel K. Lee and Jens Palsberg. Aurora: scalable sensor network simulation with precise timing. In International conference on Information processing in sensor networks, pages 477–482, 2005. [123](#)
- [Werner-Allen 2005] Geoffrey Werner-Allen, Patrick Swieskowski and Matt Welsh. MoteLab: a wireless sensor network testbed. In IEEE International Conference on Information Processing in Sensor Networks, pages 483–488, 2005. [122](#)
- [Zahn 2005] Thomas Zahn and Jochen H. Schiller. MADPastry: A DHT Substrate for Practicably Sized MANETs. In 5th Workshop on Applications and Services in Wireless Networks (ASWN2005), June 2005. [13](#)
- [Zahn 2006] Thomas Zahn and Jochen H. Schiller. DHT-based Unicast for Mobile Ad Hoc Networks. pages 179–183. Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops(PerComW’06), March 2006. [16](#)