



UNIL | Université de Lausanne

---

FACULTÉ DES HAUTES ÉTUDES COMMERCIALES  
DÉPARTEMENT DES SYSTÈMES D'INFORMATION

**Abstractions and Algorithms for Building  
Proximity-Based Mobile Applications  
in Mobile Ad Hoc Networks**

THÈSE DE DOCTORAT

présentée à la

Faculté des Hautes Etudes Commerciales  
de l'Université de Lausanne

pour l'obtention du grade de  
Docteur ès sciences en systèmes d'information

par

Behnaz BOSTANIPOUR

Directeur de thèse  
Prof. Benoît Garbinato

Jury

Prof. Olivier Cadot, Président  
Prof. Valérie Chavez, experte interne  
Prof. Pascal Felber, expert externe  
Prof. Patrick Eugster, expert externe

LAUSANNE  
2016

## IMPRIMATUR

---

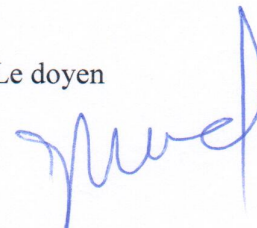
Sans se prononcer sur les opinions de l'autrice, la Faculté des Hautes Études Commerciales de l'Université de Lausanne autorise l'impression de la thèse de Madame Behnaz BOSTANIPOUR, titulaire d'un Master of Science MSc en Systèmes de communication de l'Ecole Polytechnique Fédérale de Lausanne (EPFL), en vue de l'obtention du grade de docteur ès sciences en Systèmes d'Information.

La thèse est intitulée :

### **ABSTRACTIONS AND ALGORITHMS FOR BUILDING PROXIMITY-BASED MOBILE APPLICATIONS IN MOBILE AD HOC NETWORKS**

Lausanne, le 5 septembre 2016

Le doyen



Jean-Philippe Bonardi



# Thesis Jury

## **Professor Benoît Garbinato**

Professor at the Faculty of Business and Economics of the University of Lausanne.  
Thesis supervisor.

## **Professor Olivier Cadot**

Professor at the Faculty of Business and Economics of the University of Lausanne.  
President of the Jury.

## **Professor Valérie Chavez**

Professor at the Faculty of Business and Economics of the University of Lausanne.  
Internal expert.

## **Professor Pascal Felber**

Professor at the Computer Science Department, University of Neuchâtel, Switzerland.  
External expert.

## **Professor Patrick Eugster**

Professor at the College of Science, Purdue University, West Lafayette, Indiana, USA.  
External expert.




Université de Lausanne  
Faculté des Hautes Études Commerciales

Doctorat en Systèmes d'Information

Par la présente, je certifie avoir examiné la thèse de doctorat de

**Behnaz BOSTANIPOUR**

Sa thèse remplit les exigences liées à un travail de doctorat.  
Toutes les révisions que les membres du jury et le soussigné ont demandées  
durant le colloque de thèse ont été prises en considération  
et reçoivent ici mon approbation.

Signature :  Date : 2 septembre 2016

Prof. Benoît GARBINATO  
Directeur de thèse




Université de Lausanne  
Faculté des Hautes Études Commerciales

Doctorat en Systèmes d'Information

Par la présente, je certifie avoir examiné la thèse de doctorat de

**Behnaz BOSTANIPOUR**

Sa thèse remplit les exigences liées à un travail de doctorat.  
Toutes les révisions que les membres du jury et la soussignée ont demandées  
durant le colloque de thèse ont été prises en considération  
et reçoivent ici mon approbation.

Signature :  Date : 2 septembre 2016

Prof. Valérie CHAVEZ  
Membre interne du jury





Université de Lausanne  
Faculté des Hautes Études Commerciales

Doctorat en Systèmes d'Information

Par la présente, je certifie avoir examiné la thèse de doctorat de

**Behnaz BOSTANIPOUR**

Sa thèse remplit les exigences liées à un travail de doctorat.  
Toutes les révisions que les membres du jury et le soussigné ont demandées  
durant le colloque de thèse ont été prises en considération  
et reçoivent ici mon approbation.

Signature : \_\_\_\_\_ *P. Felber* Date : 2 septembre 2016

Prof. Pascal FELBER  
Membre externe du jury



Université de Lausanne  
Faculté des Hautes Études Commerciales

Doctorat en Systèmes d'Information

Par la présente, je certifie avoir examiné la thèse de doctorat de

**Behnaz BOSTANIPOUR**

Sa thèse remplit les exigences liées à un travail de doctorat.  
Toutes les révisions que les membres du jury et le soussigné ont demandées  
durant le colloque de thèse ont été prises en considération  
et reçoivent ici mon approbation.

Signature :



Date :

2 septembre 2016

Prof. Patrick EUGSTER  
Membre externe du jury



To the memory of gardens of my youth in Shemiran  
and their magnificent tall trees, which do not exist any more.



ای نسخہ نامہ الہی کہ تویی      وی آئینہ جمال شاہی کہ تویی  
سیرون ز تو نیست هر چه در عالم هست      در خود بطلب هر آنچه خواهی کہ تویی

دیوان شمس - مولانا جلال الدین بلخی رومی

*“You are a manuscript of a divine letter.  
You are a mirror reflecting a noble face.  
This universe is not outside of you.  
Look inside yourself;  
everything that you want,  
you are already that.”*

Jalal ad-Din Balkhi Rumi, *Divan-e Shams*





# Abstract

The ubiquity of mobile devices and particularly smartphones has caused the emergence of a new trend of distributed applications known as *Proximity-Based Mobile* (PBM) applications. These applications enable a user to interact with others in a defined range and for a certain time duration for different purposes such as social networking, dating, gaming and driving. The goal of this thesis is to introduce a set of programming abstractions and algorithms that can be used for building PBM applications in a category of mobile networks, called *mobile ad hoc networks* (MANETs). In fact, the characteristics of MANETs make them a promising technology to enable PBM applications. However, the existing abstractions and algorithms in the literature of MANETs are not fully adequate for building PBM applications. Thus, in this thesis we define *proximity-based durable broadcast* and *proximity-based neighbor detection* as the main requirements of PBM applications. Then, in each part of the thesis, we introduce abstractions and algorithms which address one of these requirements.

In the first part of the thesis, we present abstractions and algorithms for proximity-based durable broadcast. Thus, we introduce *spotcast*, a new communication abstraction that enables a node to disseminate a message for a given time duration to all nodes located within a given range. We present three variants of spotcast, which differ in their timing guarantees for message delivery. We also introduce *scoped broadcast*, a communication abstraction that enables a node to disseminate a message to all nodes located within a given range. We introduce two variants of scoped broadcast: a reliable synchronous variant as well as an asynchronous variant. We discuss the implementability of each scoped broadcast variant in the single-hop and multi-hop cases. We then introduce a *generic algorithm*, which can implement the three spotcast variants using different scoped broadcast variants and different types of message buffers.

In the second part of the thesis, we present abstractions and algorithms for proximity-based neighbor detection. We begin by adapting the *hello protocols* (which are one of the most famous neighbor detection algorithms in MANETs) for *effective* and *efficient*

neighbor detection in three typical urban environments where PBM applications are mostly used. Effectiveness refers to the degree to which the detection is successful and efficiency refers to the degree to which the detection is energy saving. Based on a realistic simulation-based study, we show that in all three environments, there is a conflict between effectiveness and efficiency. Then for each environment, we propose a communication strategy which makes a good tradeoff between effectiveness and efficiency in that environment. One of the limitations of the *hello protocols* is that they only detect current neighbors. However, for some of the existing PBM applications, neighbor discovery is not restricted to current neighbors. Thus, we continue the second part of the thesis by introducing a new neighbor detection abstraction called *the time-limited neighbor detector*. This abstraction enables a node to detect its neighbors in the past, present and up to some bounded time interval in the future. We introduce two algorithms that implement the time-limited neighbor detector abstraction based on the notion of *virtual mobile nodes* already presented in the literature. The first algorithm is simple but limited and uses a single virtual mobile node. The second algorithm is more general and advanced and uses multiple virtual mobile nodes.

**Keywords**—Distributed Systems; Distributed Algorithms; Wireless Ad Hoc Networks; Mobile Ad Hoc Networks (MANETs); Mobile Computing; Proximity-Based Mobile (PBM) Applications; Smartphone; Proximity-Based Broadcast; IEEE 802.11; Energy Efficiency; Neighbor Detection; Virtual Mobile Node; Location Prediction; Log-normal Shadowing Radio Propagation Model; ns-2 Network Simulator; Wireless Network Interface Card.

# Résumé

L’omniprésence des appareils mobiles et en particulier des smartphones, a causé l’émergence d’un nouveau type des applications réparties qu’on appelle *les applications mobiles sensibles à la proximité* ou *applications PBM* (Proximity-Based Mobile). Ces applications permettent à un utilisateur d’interagir avec tous les autres utilisateurs situés dans un rayon donné autour de lui pendant une durée déterminée. Ces interactions peuvent avoir des fins différentes, telles que le réseautage social, le jeu en ligne ou le support à la conduite automobile. L’objectif de cette thèse est de proposer des abstractions et des algorithmes qui peuvent être utilisés pour construire les applications PBM dans un type de réseau mobile qu’on appelle *réseaux mobiles ad hoc* ou MANETs (Mobile Ad-hoc Networks). En effet, les caractéristiques des MANETs en font une technologie prometteuse pour les applications PBM. Toutefois, les abstractions et les algorithmes existants dans la littérature des MANETs ne sont pas tout à fait adéquats pour construire les applications PBM. Dans cette thèse, nous définissons *le broadcast durable sensible à la proximité et la détection de voisins sensible à la proximité* comme les exigences principales des applications PBM. Ensuite, dans chaque partie de la thèse, nous proposons des abstractions et des algorithmes qui sont conçus pour satisfaire ces exigences.

Dans la première partie de la thèse, nous présentons des abstractions et des algorithmes pour le broadcast durable sensible à la proximité. Pour ce faire, nous introduisons *spotcast*, une nouvelle abstraction qui permet à un noeud de diffuser un message pendant une durée déterminée à tous les noeuds situés dans un rayon donné autour de lui. Nous présentons trois variantes de *spotcast*, qui diffèrent par leurs garanties temporels pour la livraison des messages. Nous présentons aussi *scoped broadcast*, une abstraction qui permet à un noeud de diffuser un message à tous les noeuds situés dans un rayon donné autour de lui. Nous introduisons deux variantes de *scoped broadcast*: une variante synchrone fiable ainsi qu’une variante asynchrone. Nous discutons de l’implémentation single-hop et multi-hop de chaque variante de *scoped broadcast*. Ensuite, nous présentons *un algorithme générique*,

qui implémente les trois variantes de spotcast en utilisant les différentes variantes de scoped broadcast et différents types de tampon des messages.

Dans la deuxième partie de la thèse, nous présentons les abstractions et les algorithmes pour la détection de voisins sensible à la proximité. Nous commençons par adapter les protocoles *hello* (qui font partie des algorithmes de détection de voisins les plus utilisés dans les MANETs) pour la détection *efficace* et *efficace* de voisins dans trois environnements urbains typiques où les applications PBM sont utilisées. Par efficacité, nous entendons le degré auquel la détection est réussie et par efficacité, nous entendons le degré auquel la détection est efficace du point de vue de sa consommation d'énergie. En se basant sur des simulations réalistes, nous montrons que dans les trois environnements, il y a un conflit entre l'efficacité et l'efficacité. Ensuite, pour chaque environnement, nous présentons une stratégie de communication qui fait un compromis entre l'efficacité et l'efficacité dans cet environnement-là. L'une des limitations des protocoles *hello* est qu'ils ne détectent que les voisins actuels. Cependant, pour certaines applications PBM, la détection de voisins n'est pas limitée aux voisins actuels. Nous poursuivons donc la seconde partie de la thèse en introduisant une nouvelle abstraction de détection de voisins, appelée *le time-limited neighbor detector*. Cette abstraction permet à un noeud de détecter ses voisins dans le passé, le présent et jusqu'à un certain intervalle de temps borné dans le futur. Nous présentons deux algorithmes qui implémentent le time-limited neighbor detector, en utilisant *les noeuds virtuels mobiles* (la notion du noeud virtuel mobile existe déjà dans la littérature). Le premier algorithme est simple et limité et utilise un seul noeud virtuel mobile. Le second algorithme est plus général et utilise plusieurs noeuds virtuels mobiles.

**Mots clefs**—Systèmes Répartis; Algorithmes Répartis; Réseaux Sans Fil Ad Hoc; Réseaux Mobiles Ad Hoc (MANETs); Informatique Mobile; Applications Mobiles Sensibles à la Proximité ou Applications PBM; Smartphone; Broadcast Sensible à la Proximité; IEEE 802.11; Économies d'énergie; Détection de Voisins; Noeud Virtuel Mobile; Prédiction de Lieu; Modèle de Propagation Radio; Simulateur Réseau ns-2; Carte Réseau Sans Fil.

فراگیری دستگاه‌های موبایل و به ویژه تلفن‌های هوشمند موجب پیدایش گونه نوینی از برنامه‌های کاربردی توزیع شده به نام برنامه‌های کاربردی موبایل مبتنی بر نزدیکی (Proximity-Based Mobile applications) شده است. به وسیله این برنامه‌های کاربردی، یک کاربر می‌تواند با دیگر کاربران در شعاعی تعریف شده در پیرامونش و برای مدت زمانی تعریف شده ارتباط برقرار کند. این برقرای ارتباط می‌تواند برای مقاصد گوناگونی همچون شبکه‌های اجتماعی، دوست‌یابی، بازی و رانندگی صورت گیرد. هدف این پایان نامه ارائه ابسترکشن‌ها (انتزاع‌ها) و الگوریتم‌های است که بتوانند برای ساختن برنامه‌های کاربردی موبایل مبتنی بر نزدیکی در گونه‌ای از شبکه‌های موبایل به نام شبکه‌های موبایل ادهاک یا منت‌ها (MANETs) استفاده شوند. در واقع، ویژگی‌های منت‌ها به آنها توانایی این را می‌دهد که فناوری نوید بخشی برای برنامه‌های کاربردی موبایل مبتنی بر نزدیکی باشند. اما، ابسترکشن‌ها و الگوریتم‌هایی که تا کنون برای منت‌ها طراحی شده اند، کاملاً مناسب ساختن برنامه‌های کاربردی موبایل مبتنی بر نزدیکی نیستند. در این پایان نامه برادکست مداوم مبتنی بر نزدیکی و همسایه‌یابی مبتنی بر نزدیکی به عنوان نیازهای بنیادی برنامه‌های کاربردی موبایل مبتنی بر نزدیکی تعریف می‌شوند. سپس، در هر بخش از این پایان نامه ابسترکشن‌ها و الگوریتم‌های ارائه می‌شوند که یکی از این نیازها را پوشش می‌دهند.

در بخش نخست این پایان نامه، ابسترکشن‌ها و الگوریتم‌هایی برای برادکست مداوم مبتنی بر نزدیکی ارائه می‌شوند. بدینسان، ما ابسترکشنی به نام اسپاتکست (spotcast) را معرفی می‌کنیم که به وسیله آن یک گره می‌تواند پیامی را در شعاعی تعریف شده در پیرامونش و برای مدت زمانی تعریف شده پخش کند. ما سه گونه از اسپاتکست با تضمین‌های زمانی متفاوت برای دریافت پیام را ارائه می‌دهیم. همچنین، ما ابسترکشنی به نام اسکوپد برادکست (scoped broadcast) را معرفی می‌کنیم که به وسیله آن یک گره می‌تواند پیامی را در شعاعی تعریف شده در پیرامونش پخش کند. ما دو گونه از اسکوپد برادکست را ارائه می‌دهیم: یک گونه قابل اعتماد و همگام (reliable synchronous) و یک گونه ناهمگام (asynchronous). ما پیاده‌سازی هر کدام از این گونه‌ها را در حالت تک‌هاپ و چند هاپ بررسی می‌کنیم. سپس، یک الگوریتم عمومی (generic) را ارائه می‌دهیم که می‌تواند سه گونه اسپاتکست را با استفاده از گونه‌های متفاوت اسکوپد برادکست و بافرهای پیام متفاوت پیاده‌سازی کند.

در بخش دوم این پایان نامه، ابسترکشن‌ها و الگوریتم‌هایی برای همسایه‌یابی مبتنی بر نزدیکی ارائه می‌شوند. ما این بخش را با تطبیق پروتکل‌های هلو (hello protocols)، که یکی از شناخته‌ترین الگوریتم‌های موجود برای همسایه‌یابی در منت‌ها هستند، آغاز می‌کنیم. به بیان دقیق‌تر، ما پروتکل‌های هلو را برای همسایه‌یابی اثر بخش و بهره‌ور در سه محیط شهری که برنامه‌های کاربردی موبایل مبتنی بر نزدیکی بیشتر در آنها استفاده

می‌شوند، تطبیق می‌دهیم. در اینجا اثر بخشی به مفهوم درجه موفقیت همسایه یابی و بهره‌وری به مفهوم درجه صرفه جویی در مصرف انرژی در فرایند همسایه یابی می‌باشد. ما بر اساس شبیه سازی‌های شبکه‌ای واقع‌گرایانه نشان می‌دهیم که در هر سه محیط، اثر بخشی و بهره‌وری در مغایرت با هم قرار دارند. سپس، برای هر محیط یک استراتژی ارتباطی پیشنهاد می‌کنیم که بده بستان (tradeoff) خوبی میان اثر بخشی و بهره‌وری برقرار می‌کند. یکی از کاستی‌های پروتکل‌های هلو این است که آنها تنها می‌توانند همسایگان کنونی را تشخیص دهند. این در حالی است که در برخی برنامه‌های کاربردی موبایل مبتنی بر نزدیکی، همسایه یابی مختص به یافتن همسایگان کنونی نمی‌شود. بنابراین، ما بخش دوم این پایان نامه را با ارائه یک ابسترکشن نوین برای همسایه یابی به نام همسایه یاب محدود در زمان (time-limited neighbor detector) ادامه می‌دهیم. به وسیله این ابسترکشن، یک گره می‌تواند همسایگان‌ش را در گذشته، حال و تا بازه زمانی محدودی در آینده، شناسائی کند. ما دو الگوریتم را معرفی می‌کنیم که همسایه یاب محدود در زمان را با استفاده از گره‌های مجازی متحرک پیاده سازی می‌کنند. گره مجازی متحرک (virtual mobile node) یک ابسترکشن است که پیشتر برای منت‌ها طراحی شده است. الگوریتم نخست، ساده و محدود است و همسایه یاب محدود در زمان را با استفاده از یک گره مجازی متحرک پیاده سازی می‌کند. الگوریتم دوم، پیشرفته تر و عمومی تر است و همسایه یاب محدود در زمان را با استفاده از چندین گره مجازی متحرک پیاده سازی می‌کند.

**واژگان کلیدی** - سیستم‌های توزیع شده؛ الگوریتم‌های توزیع شده؛ شبکه‌های ادهاک بی سیم؛ شبکه‌های موبایل ادهاک یا منت‌ها؛ رایانش موبایل؛ برنامه‌های کاربردی موبایل مبتنی بر نزدیکی؛ تلفن هوشمند؛ برادکست مبتنی بر نزدیکی؛ آی‌تریپل‌ئی ۸۰۲/۱۱؛ بهره‌وری انرژی؛ همسایه یابی؛ گره مجازی متحرک؛ پیشینی مکان؛ مدل انتشار رادیویی لاگ نرمال؛ شبیه ساز شبکه ان اس-۲؛ کارت شبکه بی سیم.

# Acknowledgements

I would like to thank my advisor, **Professor Benoît Garbinato**, for his guidance and his indispensable advice throughout this thesis. In particular, he taught me the value of precision and clear communication of ideas and showed me how a solution to a problem *should be made as simple as possible, but not simpler*. Moreover, I am deeply grateful to him for his constant support and his tremendous efforts to ensure the completion of this thesis under the finest conditions.

I would also like to thank the other members of my thesis jury, **Professor Pascal Felber**, **Professor Patrick Eugster**, **Professor Valérie Chavez** and **Professor Olivier Cadot**. They have provided valuable feedback and interesting perspectives on the ideas contained in this thesis.

I am grateful to my former professors and advisers at the school of computer and communication sciences at EPFL. In particular, I would like to thank **Professor Rachid Guerraoui** for his excellent course of Distributed Algorithms, which not only provided me with the basic knowledge in theoretical distributed computing but also increased my interest in designing distributed algorithms. I also thank **Professor Martin Hasler**, **Dr. Thanasis G. Papaioannou** and **Professor Karl Aberer**, who supported my decision to pursue a PhD.

Through the years of completing this research at the Information Systems Department of the university of Lausanne, I had the pleasure to get to know many great people, in particular, **Professor Christine Legner**, **Professor Yves Pigneur**, **Professor Periklis Andritsos**, **Professor Jacques Duparc** and **Professor Thibault Estier**. Special thanks to **Elisabeth Fournier Pulfer**, for being the most efficient and organized secretary and one of the most kind-hearted people that I have ever known. Furthermore, I would like to thank my former and current colleagues and

office-mates at the Distributed Object Programming (DOP) Laboratory, including **Shabnam, Adrian, François, Arielle, Bertil, Vincent** and **Vaibhav**.

The work presented in this thesis was partly funded by **the Swiss National Science Foundation** in the context of Project 200021-140762. I thankfully acknowledge this support.

Last but not least, I would like to thank God and my family. I thank God for everything, in particular, for giving me the strength to overcome obstacles and for helping me in the most amazing ways. I thank my uncles who were my first inspirations to study engineering. I would like to thank my late father who was not only a great neurologist but also a great reader of various types of books. From him, I learned to be more curious and eager to learn and to never give up on my dreams. I need to thank my late grandmother who always believed in me and encouraged me. I am grateful to my brother for being my first best friend and being there for me whenever I needed him. Finally, I thank the most important contributor of this thesis, my mother who is truly the smartest, the wisest and the most courageous person that I have ever known. Without her love, support and encouragement, none of this would have been possible.



# Contents

<b>Abstract (English/French/Persian)</b> .....	xvii
<b>Acknowledgements</b> .....	xxiii
<b>Table of Contents</b> .....	xxv
<b>List of Figures</b> .....	xxx
<b>List of Tables</b> .....	xxxii
<b>1 Introduction</b> .....	1
1.1 Proximity-Based Mobile (PBM) Applications and Mobile Ad Hoc Networks (MANETs) .....	3
1.1.1 Proximity-Based Mobile (PBM) Applications .....	3
1.1.2 Mobile Ad Hoc Networks (MANETs) .....	5
1.1.3 Why Building Proximity-Based Mobile (PBM) Applications in Mobile Ad Hoc Networks (MANETs)? .....	5
1.2 Related Work .....	6
1.2.1 Communication Abstractions and Algorithms .....	7
1.2.1.1 Traditional Communication Abstractions and Algorithms	7
1.2.1.2 Location-Aware and/or Time-Aware Communication Abstractions and Algorithms .....	12
1.2.2 Neighbor Detection Abstractions and Algorithms .....	13
1.3 Problem Statement and Research Questions .....	15
1.4 Thesis Overview .....	19
1.5 List of Publications .....	23
References .....	24

## Part I Abstractions and Algorithms for Proximity-Based Durable Broadcast

<b>2 Spotcast – A Communication Abstraction for Proximity-Based Mobile Applications</b> . . . . .	33
2.1 Distributed Systems: Evolution or Revolution? . . . . .	33
2.1.1 First Evolution . . . . .	34
2.1.2 Then Paradigm Shift . . . . .	34
2.1.2.1 On-the-spot Survey . . . . .	35
2.1.2.2 Social Radar . . . . .	35
2.1.2.3 Mobile Photo Sharing . . . . .	35
2.1.3 Spotcast: A New Communication Abstraction . . . . .	36
2.1.4 Roadmap . . . . .	36
2.2 System Model . . . . .	36
2.3 The Spotcast Abstraction . . . . .	37
2.3.1 Core Spotcast Properties . . . . .	38
2.3.2 Spotcast Variants and their Validity Properties . . . . .	38
2.3.2.1 Timely Spotcast . . . . .	38
2.3.2.2 Eventual Spotcast . . . . .	39
2.3.2.3 Exhaustive Spotcast . . . . .	40
2.4 A Spotcast Algorithm . . . . .	40
2.4.1 Global Positioning Service . . . . .	41
2.4.2 Scoped Broadcast Service . . . . .	41
2.4.3 Generic Spotcast Algorithm . . . . .	42
2.4.4 Correctness and Implementability . . . . .	44
2.4.4.1 Validity of Timely Spotcast . . . . .	44
2.4.4.2 Validity of Eventual Spotcast . . . . .	45
2.4.4.3 Validity of Exhaustive Spotcast . . . . .	46
2.5 Implementability of Scoped Broadcast Service . . . . .	46
2.5.1 Implementability of the Fair-Loss Delivery Property . . . . .	47
2.5.1.1 Single-hop Case . . . . .	47
2.5.1.2 Multi-hop Case . . . . .	47
2.5.2 Implementability of the Timely Delivery Property . . . . .	48
2.5.2.1 Single-hop Case . . . . .	48
2.5.2.2 Multi-hop Case . . . . .	49

2.6 Related Work .....	49
2.6.1 Geocast .....	49
2.6.2 Mobicast .....	50
2.6.3 Location-Based Publish/Subscribe .....	51
2.7 Conclusion .....	51
References .....	52

## Part II Abstractions and Algorithms for Proximity-Based Neighbor Detection

<b>3 Effective and Efficient Neighbor Detection for Proximity-Based Mobile Applications .....</b>	<b>57</b>
3.1 Introduction .....	58
3.1.1 Contributions and Roadmap .....	60
3.2 System Model .....	61
3.2.1 Processes .....	61
3.2.2 Time .....	61
3.2.3 Communication .....	61
3.2.4 Environment .....	62
3.2.5 Neighbor Detection Algorithm .....	63
3.3 Problem Statement .....	63
3.4 Evaluation of Effectiveness .....	65
3.4.1 Approach to Calculate the Neighbor Detection Probability .....	65
3.4.2 Simulation Setup .....	66
3.4.3 Results .....	69
3.4.3.1 Packet Dropping Metrics .....	70
3.4.3.2 Metrics-based Interpretation of the Results .....	72
3.4.3.3 Impact of Changing $pow_{tx}$ and $\Delta_{period}$ on Neighbor Detection Probability .....	74
3.4.3.4 The Most Effective Strategy .....	75
3.5 Evaluation of Efficiency .....	76
3.5.1 Energy Consumption Model .....	77
3.5.2 Energy Consumption Calculation Algorithm .....	78
3.5.3 Results .....	82

3.5.3.1 Impact of Changing $pow_{tx}$ and $\Delta_{period}$ on Energy Consumption . . . . .	83
3.5.3.2 The Most Efficient Strategy . . . . .	85
3.6 Effectiveness-Efficiency Tradeoff . . . . .	86
3.6.1 Conflict . . . . .	87
3.6.2 Approach to Make the Tradeoff . . . . .	88
3.6.3 Benefit–Cost Ratio (BCR) . . . . .	89
3.6.4 Impact of changing $pow_{tx}$ and $\Delta_{period}$ on BCR . . . . .	89
3.6.5 The Tradeoff Strategy . . . . .	91
3.6.6 How Effective and Efficient is the Tradeoff Strategy? . . . . .	91
3.7 Related Work . . . . .	93
3.7.1 Enhancements of the hello Protocol . . . . .	93
3.7.2 Beacon Broadcast for VANET Safety Applications . . . . .	95
3.8 Conclusion . . . . .	97
References . . . . .	98

<b>4 Using Virtual Mobile Nodes for Neighbor Detection in Proximity-Based Mobile Applications . . . . .</b>	<b>103</b>
4.1 Introduction . . . . .	104
4.2 System Model and Definitions . . . . .	105
4.2.1 Definitions . . . . .	106
4.2.2 Timely Scoped Broadcast Service . . . . .	106
4.2.3 Global Positioning Service . . . . .	107
4.2.4 Mobility Predictor Service . . . . .	107
4.3 The Neighbor Detector Abstraction . . . . .	108
4.3.1 Neighbor Detector Variants . . . . .	108
4.3.1.1 Perfect Neighbor Detector . . . . .	108
4.3.1.2 Time-limited Neighbor Detector . . . . .	109
4.4 Implementing The Time-Limited Neighbor Detector . . . . .	109
4.4.1 Virtual Mobile Node (VMN) . . . . .	110
4.4.2 Adding a VMN to the System Model . . . . .	110
4.4.3 A Time-limited Neighbor Detector Algorithm . . . . .	112
4.4.4 Proof of Correctness . . . . .	114
4.5 Related Work . . . . .	119
4.6 Conclusion . . . . .	120

References .....	121
<b>5 A Neighbor Detection Algorithm Based on Multiple Virtual Mobile Nodes for Mobile Ad Hoc Networks .....</b>	<b>125</b>
5.1 Introduction .....	126
5.2 System Model and Definitions .....	129
5.2.1 Definitions .....	130
5.2.2 LocalCast Service .....	130
5.2.3 Global Positioning Service .....	131
5.2.4 Mobility Predictor Service .....	131
5.3 The Neighbor Detector Service .....	133
5.3.1 Neighbor Detector Variants .....	133
5.3.1.1 Perfect Neighbor Detector .....	133
5.3.1.2 Time-limited Neighbor Detector .....	134
5.4 Implementing The Time-Limited Neighbor Detector .....	135
5.4.1 Virtual Mobile Node .....	136
5.4.2 Adding Virtual Mobile Nodes to the System Model .....	137
5.4.3 The Scan Path of a Virtual Mobile Node .....	139
5.4.4 Neighbor Detector Algorithm .....	142
5.4.5 Proof of Correctness .....	145
5.4.5.1 Intuition behind the Proof .....	146
5.4.5.2 Preliminaries .....	147
5.4.5.3 The Proof .....	153
5.4.6 Impact of Increasing the Number of Virtual Mobile Nodes on $\Delta_{predict}^{min}$ .....	162
5.5 Performance Discussion .....	164
5.5.1 Scalability with respect to the Number of Virtual Mobile Nodes .....	164
5.5.2 Performance Optimization .....	165
5.5.2.1 Optimizing the Implementation of the Virtual Mobile Nodes .....	166
5.5.2.2 Optimizing the Neighbor Detector Algorithm .....	167
5.6 Related Work .....	168
5.6.1 Neighbor Detection Algorithms .....	169
5.6.2 Mobility-assisted Algorithms .....	170
5.6.3 Virtual Mobile Node-based Algorithms .....	174

5.7 Conclusion .....	174
5.A Appendix: Finding an Upper Bound for the Scan Path Length of a Virtual Mobile Node .....	176
References .....	178
<b>6 Conclusion .....</b>	<b>183</b>
6.1 Contributions .....	183
6.2 Future Work .....	186
References .....	188

# List of Figures

2.1	Validity Properties of Spotcast Variants. . . . .	39
2.2	Spotcast – Architecture Overview. . . . .	40
3.1	Simulation Map . . . . .	67
3.2	Same strategy in different environments. The vertical error bars present the standard deviation for the detection probability . . . . .	71
3.3	Values of packet dropping metrics for different strategies in the indoor with hard partitions environment . . . . .	71
3.4	Impact of increasing $pow_{tx}$ on the detection probability in different environments . . . . .	71
3.5	Impact of decreasing $\Delta_{period}$ on the detection probability in different environments . . . . .	71
3.6	Strategies ranked in the descending order with respect to their corresponding detection probability in different environments . . . . .	76
3.7	Variation of $E_{active}$ and $E_{passive}$ over different strategies when $T_{total} = \Delta_{neighborhood} = 4$ seconds . . . . .	84
3.8	Variation of $E_{active}$ and $E_{passive}$ over different strategies for high values of $T_{total}$ (different hours) . . . . .	84
3.9	Impact of increasing $pow_{tx}$ and decreasing $\Delta_{period}$ on the energy consumption . . . . .	84
3.10	Strategies ranked in the ascending order with respect to their corresponding energy consumption . . . . .	86
3.11	Impact of increasing $pow_{tx}$ on BCR in different environments . . . . .	90
3.12	Impact of decreasing $\Delta_{period}$ on BCR in different environments . . . . .	90

3.13	Strategies ranked in the descending order with respect to their corresponding BCR in different environments . . . . .	90
4.1	VMN scans the region $R$ . . . . .	111
4.2	The worst case scenario . . . . .	115
5.1	The subfigures correspond to the case where the number of virtual mobile nodes (denoted by $n$ ) is equal to four. (a) Disk $R$ is presented where the grey area corresponds to a subregion $R_i$ . (b) Each virtual mobile node scans its associated subregion in the form of collect and distribute scans. The arrows indicate the direction of motion. . . . .	138
5.2	Hexagonal tessellation of the surface of a subregion $R_i$ . Each hexagon approximates a circle of radius $r_{com}$ and hence its circumradius is equal to $r_{com}$ . In the figure, we present the circle and its radius in red only for one hexagon. . . . .	140
5.3	$t_{b,k}$ and $t_{e,k}$ of $round_k$ . . . . .	147
5.4	Examples of occurrence of two consecutive atomic phases $\phi_i$ and $\phi_j$ in a nice execution. The cases correspond to the cases of the proof of Lemma 5.6. . . . .	151



# List of Tables

2.1	Spotcast – Correctness and Implementability.....	45
3.1	Values of LNS Parameters for each Environment. ....	62
3.2	Effectiveness rank and ratio of the strategies in different environments	76
3.3	Efficiency rank and ratio of the strategies .....	86
3.4	Comparison of the most effective strategy, the most efficient strategy and the tradeoff strategy in different environments .....	92
3.5	Impact of changing $pow_{tx}$ or $\Delta_{period}$ on effectiveness, efficiency and BCR in different environments .....	92



# Chapter 1

## Introduction

A distributed application is a software that executes on a collection of autonomous networked computers and which aims at producing some sort of cooperation. Abstractions and algorithms form the building blocks of a distributed application. An abstraction is an artifact that provides a functional interface and a set of guarantees for an entity or a service in a distributed application. An algorithm is either an implementation of an abstraction (i.e., it satisfies the abstraction's guarantees) or is *best-effort*, i.e., it implements no abstraction and satisfies some fair but not formal guarantees.

Until the mid-nineties, distributed applications tended to be strictly confined to intranets, that is, networks within organizations such as banks or companies. These networks were usually composed of wired stationary computers and protected by a firewall from the outside world. However, due to the exponential increase in the number of computers of all sizes interconnected via the Internet, distributed applications soon broke free from intranets and evolved. The emergence of peer-to-peer file sharing applications such as Kazaa in late nineties and the growing interest for cloud computing in the recent years are famous examples of this evolution.

Today with the increasing adoption and usage of mobile devices, in particular smartphones, we face the emergence of a new blend of distributed applications, known as *proximity-based mobile* (PBM) applications. These applications enable a user to interact with others in a defined range and for a certain time duration e.g., for social networking (WhosHere [73], LoKast [52], iGroups [35], LocoPing [51]), gaming (local multiplayer apps [50]) and driving (Waze [72]). Compared to typical distributed applications that flourished in the past two decades, PBM applications have intrinsic *spatio-temporal* semantics, i.e., they are defined for a given range and some time duration.

In this thesis, we consider the problem of building PBM applications in a category of mobile networks called *mobile ad hoc networks* (MANETs). As we further discuss in this chapter, the characteristics of MANETs make them a promising technology to enable PBM applications. However, the existing abstractions and algorithms in the literature of MANETs are not fully adequate for building PBM applications. In fact, the majority of the existing abstractions and algorithms in the literature of MANETs are originally devised for traditional (or intranet-oriented) applications. Accordingly, these abstractions and algorithms are location and time-oblivious. Moreover, the main concern of these abstractions and algorithms is usually to reach some sort of network-wide information consistency, whereas the main concern of PBM applications is the *spatio-temporal* nature of interactions i.e., the fact that they are defined for a certain range and a certain time duration. In the literature of MANETs, there also exist some location and/or time-aware abstractions and algorithms. However, they cannot entirely satisfy the requirements of PBM applications.

To fill the described research gap, in this thesis we design and develop new abstractions and algorithms for building PBM applications in MANETs. We also propose ways to adapt some of the existing algorithms in order to be used as the building blocks for PBM applications.

The rest of this chapter is structured as follows. In Section 1.1, we introduce the definition of PBM applications and present their examples and requirements. We then present the definition of MANETs. Finally, we discuss the reason behind our choice of MANETs as the underlying technology for building PBM applications. In Section 1.2, we review the literature of MANETs and show that the existing abstractions and algorithms are not fully adequate for building PBM applications. In Section 1.3, we present the overall question that we address in this thesis as well as the research questions which are derived from it. In Section 1.4, we present an overview of the thesis and its chapters. In particular, we define how each chapter addresses some of the research questions already presented in Section 1.3. Finally, in Section 1.5 we present the list of publications related to this thesis.

## 1.1 Proximity-Based Mobile (PBM) Applications and Mobile Ad Hoc Networks (MANETs)

Throughout this thesis, we focus on designing and developing abstractions and algorithms that can be used for building proximity-based mobile (PBM) applications in mobile ad hoc networks (MANETs). Thus, in this section we first introduce the definition of PBM applications and present their examples and requirements. We then present the definition of mobile ad hoc networks. Finally, we discuss the reason behind our choice of MANETs as the underlying technology for building PBM applications.

### 1.1.1 Proximity-Based Mobile (PBM) Applications

**Definition.** We define a proximity-based mobile (PBM) application, as a distributed application which has three characteristics:

- (1) it involves a set of mobile devices;
- (2) interaction between devices follow a peer-to-peer communication model;
- (3) it has *spatio-temporal* semantics, i.e., it is defined for a given range (proximity of the devices) and a certain time duration.

The most common examples of PBM applications can be found in some of existing smartphone applications. These applications are used for different purposes such as social networking or dating (WhosHere [73], LoKast [52], iGroups [35], LocoPing [51], FireChat [23]), event organizing and professional networking (Bizzabo [7]), gaming (local multiplayer apps [50]) and driving (Waze [72]).

**Requirements.** Here we list the requirements of PBM applications, which are mainly defined based on the characteristics of current PBM applications.

- *Proximity-Based Durable Broadcast.* In various existing PBM applications, a message should be sent to all nearby devices for some time duration. An example of such applications is Waze [72]. Waze is a smartphone-based driving application which among other things, enables users to create real-time and accurate news about conditions of routes and share them with others in their proximity. For

instance, if a user Alice is stuck in a traffic jam, she can report this event for a certain amount of time in her proximity, so other users approaching her location can be notified and can avoid the traffic jam. Another example is LoKast [52]. LoKast is a smartphone-based social networking application which among other things, enables a user Alice, participating in some social event, to share photos taken during the event with other users attending that event.

The tasks described in both above mentioned applications, can be performed by a proximity-based durable broadcast: in the case of Waze, the traffic alert can be broadcast in the proximity of Alice for some time duration and in the case of LoKast, the URL pointing to the photos taken at the event can be broadcast in the proximity of Alice for some time duration. However, the message delivery guarantees of proximity-based durable broadcast are not the same in both applications. More precisely, in the case of Waze it is important to receive the traffic alert quickly and in a time-limited interval whereas in the case of LoKast, a user Bob who was in the proximity of Alice during the event, can receive the URL even after the event, i.e., as soon as he stays in the neighborhood of Alice long enough (to deliver the URL).

- *Proximity-Based Neighbor Detection.* Discovering who is nearby is one of the basic requirements of PBM applications. It is the preliminary step for further interactions between users. It also enables users to extend their social network from the people that they know to the people that they might not know but who are in their proximity. For instance, in a simple usage scenario of smartphone-based social networking applications such as WhosHere [73] or LoKast [52], a user Alice first discovers nearby users and then decides to view their profiles, add them as friends or start a chat with a user or a group of users with her smartphone. She will be constantly notified about the changes in her neighborhood, i.e., whenever a user enters or leaves her proximity.

The neighbor discovery is not always limited to the current neighbors. For instance, with the smartphone-based social networking applications such as iGroups [35] or LocoPing [51], Alice can discover others who were in her vicinity during a past event (e.g., concert, tradeshow, wedding) or simply during a past time interval (e.g., the past 24 hours). One can also think of applications that provide Alice with the list of people who will be in her proximity up to some time interval in the future and thus create the potential for new types of social interactions.

Based on the above mentioned examples, a proximity-based neighbor detection should have the following characteristics:

- (1) It should be *effective*: even if a device remains in proximity for a limited amount of time, it should be detected with a high probability as a neighbor;
- (2) It should be *efficient*: due to the limited battery life of mobile devices, the neighbor-detection should be performed by consuming as little energy as possible;
- (3) It should detect not only the current neighbors, but also the past and the future neighbors (up to some bounded time-interval).

### 1.1.2 Mobile Ad Hoc Networks (MANETs)

Mobile ad hoc networks are self-configuring, infrastructureless networks of mobile nodes (devices). In a MANET, each node is equipped with wireless communication and has a *transmission range*. The transmission range of a node is defined by the transmission power and the frequency of its radio and is influenced by the environmental factors such as radio interference, packet collision, obstructions and movement. Thus, in idealized conditions (i.e., in the absence of any interference, obstruction, movement, etc...), when a node sends a message using its radio every node which is in its transmission range will receive the message. This is known as a broadcast at the *medium access control (MAC)* layer and the nodes that receive the message are *one hop* away from the sender. If a node is outside the transmission range of the sender and should receive the message, intermediate nodes have to route the message. This is known as *multi-hop communication* [70].

### 1.1.3 Why Building Proximity-Based Mobile (PBM) Applications in Mobile Ad Hoc Networks (MANETs)?

In this thesis we consider an underlying MANET architecture to build PBM applications. We acknowledge the fact that, currently the majority of PBM applications use infrastructure-based architectures and MANETs are mostly used to enable military and disaster relief applications. Thus, we do not dismiss the idea of using infrastructure-based architectures to build PBM applications. However, we believe

that MANETs are a promising technology to enable PBM applications for the two following reasons:

- (1) Characteristics of MANETs make them a natural technology for building PBM applications. More precisely, similar to PBM applications, MANETs have a *spatio-temporal* nature i.e., two nodes can communicate in MANETs if they are in certain distance to each other (to have radio connectivity) for a certain amount of time. Moreover, MANETs operate in a fully distributed fashion without the aid of a central authority. Therefore, they are preferable when a fast and unpredicted communication needs to take place between nodes that are in proximity of each other for a certain amount of time.
- (2) There is a good chance that the usage of MANETs in urban environments (i.e., where PBM applications are mostly used) will grow in the future. In fact, mobile devices are increasingly equipped with ad hoc communication capabilities (e.g., WiFi in ad hoc mode or Bluetooth). Moreover, MANETs can be used in urban environments as a complement to cellular networks (e.g., in a hybrid architecture) to solve problems such as poor coverage of base stations on the edge of the cells. Accordingly, in recent years some PBM applications have been proposed that can be executed in hybrid and/or pure ad hoc modes. An example of such applications is FireChat [23]. FireChat is a smartphone-based social networking application that was released on the App store in 2014 and can be executed in both cellular and ad hoc networks. Another example is iGropes [35], a smartphone-based social networking application that can be executed in hybrid architectures.

## 1.2 Related Work

In Section 1.1, we defined *proximity-based durable broadcast* and *proximity-based neighbor detection* as the two main requirements of PBM applications. Thus, in this section, we first survey the existing communication abstractions and algorithms in MANETs and discuss the possibility of using them for proximity-based durable broadcast. We then survey the existing neighbor detection abstractions and algorithms in MANETs and discuss the possibility of using them for proximity-based neighbor detection.



## 1.2.1 Communication Abstractions and Algorithms

In the following, we discuss two main categories of communication abstractions and algorithms in MANETs: *the traditional communication abstractions and algorithms* that are time and location-oblivious and *the time and/or location-aware communication abstractions and algorithms*.

### 1.2.1.1 Traditional Communication Abstractions and Algorithms

The traditional communication abstractions and algorithms are originally designed for the traditional distributed systems such as intranets and wired stationary networks. Here, we present a review of the most famous classes of the traditional communication abstractions and algorithms namely, *Broadcast*, *Unicast*, *Multicast* and *Consensus*. For each class, we study its most notable implementations in MANETs. As we describe, the implementation of these abstractions and algorithms in MANETs causes some difficulties, which are due to their original design. Finally, in a *discussion* section, we summarize the results of our review and discuss the possibility of using these abstractions and algorithms for proximity-based durable broadcast.

**Broadcast.** The *broadcast* problem refers to the sending of a message by a source process, called the broadcaster, to all processes in the system. There exist several variants of the broadcast abstraction [28]. They mainly differ in their message delivery and ordering guarantees. These guarantees are defined assuming a traditional wired network model. Thereby, implementing them in MANETs causes difficulties. For instance, one of the basic guarantees offered by many broadcast variants is *reliability*. A broadcast is said to be reliable if all correct processes in the system deliver the broadcast message. In MANETs, guaranteeing reliability for a broadcast is challenging. Message loss in MANETs is more frequent than in traditional networks because of collisions and radio interferences. Node mobility also results in a frequently changing network topology that makes using stable structures such as trees for broadcasting more critical. In addition, the broadcast message should often be retransmitted by some nodes in order to reach all the nodes in the network. However, retransmission is expensive (in terms of bandwidth and energy consumption) and should be avoided if it is possible. Because of these reasons, several papers assume that achieving full reliability for MANETs is impractical [53]. Therefore, they consider reliable broadcasting in MANETs as the problem of minimizing the

number of forwarding nodes (to optimize energy and bandwidth), while maximizing the number of nodes that deliver the message (to increase reliability). Some of the proposed algorithms for reliable broadcasting in MANETs use a type of contextual information to choose the forwarding nodes. The contextual information enables these algorithms to better deal with the constantly changing network topology and mobility of nodes [27].

**Unicast, Multicast.** *Unicast* refers to sending a message from a source to a destination process. *Multicast* is the transmission of packets to a group of zero or more hosts often identified by a single destination address. In MANETs, both unicast and multicast are often performed in a multi-hop manner, since the radio transmission range of the source node does not necessarily cover all the destinations. Multicast routing tries to avoid multiple transmissions of the same message to receivers belonging to the same subset. Thus, in applications where subsets of nodes require the same specific information, multicast is more resource efficient and consequently preferable to unicast.

The routing problem in MANETs is different compared to wired networks. Firstly, the routing is more difficult since the network topology changes frequently as a result of node movement. Secondly, the resources such as bandwidth and devices battery power are more limited. Therefore, the routing protocols developed for wired networks are not usually suitable for MANETs. For instance, Distance Vector routing and Link State routing are two of the most popular dynamic routing algorithms used in wired networks [69]. These algorithms assume predictable network properties, such as static link quality and network topology. In MANETs, the network topology changes frequently and the link quality is variable. Thus, in these networks, using a Distance Vector or a Link State routing protocol designed for wired networks results to a significant increase of the control overhead. The increase in the control overhead may even overuse the limited network bandwidth. Routing information inconsistency and route loops are other problems caused by using a Distance Vector or a Link State routing algorithm designed for wired networks in MANETs. Multicast routing, in particular, is difficult in MANETs. Node mobility excludes the use of a fixed multicast topology and makes keeping track of the multicast group membership more complicated than in wired networks.

In the literature, various unicast and multicast routing protocols have been proposed for MANETs. However, there exists no unique routing solution that works well

in all scenarios with different traffic overloads, link capacities, connectivity, network sizes and node mobility models [49; 1].

One way to classify MANET-routing protocols is to consider the mechanism that they use to acquire and update the routing information. In a *proactive routing* protocol, nodes keep a routing table with the next hop of every possible packet destination. Here, the challenge is to keep an up-to-date routing table given the frequent changes in network topology. Wireless Routing Protocol (WRP) [59], the Destination Sequence Distance Vector (DSDV) [62] and the Fisheye State Routing (FSR) are examples of proactive mobile ad hoc network routing protocols.

A *reactive routing* protocol discovers routes only when requested by the application (i.e. *on demand*). In MANETs, reactive routing protocols have better scalability than proactive routing protocols, since they result in less control overhead. However, using reactive routing protocols can cause the source nodes to experience long delays for route searching before they can send their data packets. Dynamic Source Routing (DSR) [37] and Ad hoc On-demand Distance Vector routing (AODV) [63] are examples for reactive routing protocols for mobile ad hoc networks.

*Hybrid routing* protocols combine both proactive and reactive routing to reduce the drawbacks of both approaches. Hybrid routing usually relies on a hierarchical network architecture where proactive and reactive routing are applied to different levels of the hierarchy. Zone Routing Protocol (ZRP) [29] and Hybrid Ad hoc Routing Protocol (HARP) [60] are examples of hybrid routing protocols designed for MANETs.

Contextual information such as node location is also exploited by some MANET routing protocols. In a *location-based routing* protocol, the locations of nodes are used by routing protocols to better deal with the highly dynamic environments of MANETs. Location Aided Routing (LAR) [41] and Distance Routing Effect Algorithm for Mobility (DREAM) [4] are examples of location-based routing protocols designed for MANETs.

**Consensus.** In the *consensus* problem, processes in a system have to agree on a common and irrevocable value called *the decision value*, which is the initial value of one of the processes.

Consensus is a fundamental building block for many distributed applications. For instance, it can be used to implement *atomic broadcast* [16; 28]. In fact consensus and atomic broadcast are equivalent from a solvability point of view. Atomic broadcast guarantees that all processes deliver the same set of messages in a common global

order. One of the applications of atomic broadcast is to build *fault-tolerant systems*. A fault tolerant system is a system which offers *availability* and *reliability*. Availability ensures that the system is ready to be used immediately. Reliability ensures that the system can continue to run without failures. A service can be made fault-tolerant by replicating its state to a group of servers, so that even if some servers fail, the service remains available. Replication management can be ensured by the *state machine replication approaches* [43; 44; 65; 66]. In these approaches, to maintain replica consistency, processes should receive and process the same sequence of requests. This is achieved by using atomic broadcast [28; 9].

It is preferable to consider the consensus problem in MANETs as the consensus problem in an asynchronous system. In fact, an asynchronous system model seems to be more suitable for MANETs, where message latencies are usually unpredictable. However, according to the *FLP impossibility* result [24], consensus cannot be solved deterministically in an asynchronous system that is subject to even a single crash failure. Intuitively, this result stems from the fact that in asynchronous systems it is impossible to distinguish between a process crash and a process which is very slow.

To overcome this impossibility, some solutions were proposed in the literature. One solution is to assume that the asynchronous system eventually becomes synchronous (*partial synchrony*) [21; 17]. Another solution is to probabilistically guarantee the termination property of the consensus by using a random number generator (*randomization*) [5; 22]. Chandra and Toueg also proposed another solution by augmenting the asynchronous system with unreliable *failure detectors* [10]. In this solution, a system model is considered in which processes can crash but links are reliable. This model was originally assumed in [24] and was later used (with extensions in some cases) by a number of papers addressing the consensus problem. In the traditional systems achieving link reliability is not usually hard. However, in MANETs, link failures are more frequent than the traditional networks and achieving reliability is expensive. Therefore not only process failures but also link failures should be considered for solving consensus in these networks [9].

In [18; 61], the consensus problem is considered in the presence of message losses. To overcome message losses, each process periodically resends the latest message that it has sent and a process skips the current round if it receives a message from a higher round. These approaches require high message complexity and seem not to be very suitable for MANETs.

In [13], the consensus problem is considered for wireless ad hoc networks with locally unknown participants and crash-stop model. The network is divided into a series of non-overlapping grid squares. Every node knows *a priori* its location in the grid and every grid square is assumed to be populated. First, single-hop consensus is run for each grid square and then all nodes gossip the local decisions. Once a node receives the values for each grid square decision, it can decide by applying a deterministic function on the set of received values. The paper assumes that inter-node communication is synchronous and nodes do not crash in the middle of executing a broadcast. The various assumptions taken by the protocol seem to make it restrictive for MANETs.

In [74], a consensus protocol for MANETs based on  $\diamond P$  failure detector [26] is proposed. A two-layer hierarchy on the network is imposed by clustering and  $k$  predefined nodes are chosen as clusterheads such that  $k < n$  where  $n$  denotes the number of nodes in the network. Each mobile node is associated with a clusterhead. The protocol tolerates  $f$  faulty nodes and requires one correct clusterhead. Thus,  $f < \min(k, n/2)$ . If clusterheads change during the execution, another consensus should be solved to agree on the clusterheads, which leads to circularity. Authors also state that their solution can be used with lossy links if it undergoes complicated design changes [9].

**Discussion.** As described in this section, there exist various implementations of the traditional communication abstractions and algorithms in MANETs. However, we believe that these abstractions and algorithms are not fully adequate for proximity-based durable broadcast for two main reasons. Firstly, these abstractions and algorithms are location and time-oblivious, whereas proximity-based durable broadcast is a location-aware and time-aware communication. Secondly, traditional communication abstractions and algorithms usually aim to reach some sort of network-wide information consistency such as *reliability* for message broadcast or *agreement* for consensus, whereas in the case of proximity-based durable broadcast the main concern is to disseminate a message to nodes which are in a given range during a given time period. As we have discussed, the contextual information such as location of nodes is used in some of existing implementations of traditional communication abstractions and algorithms in MANETs. Note however that the purpose of this usage is only to improve the performance in highly dynamic environment of MANETs and is not related to the semantics of these abstractions and algorithms.

### 1.2.1.2 Location-Aware and/or Time-Aware Communication Abstractions and Algorithms

We present a review of the most famous classes of location-aware and/or time-aware communication abstractions and algorithms namely, *Geocast*, *Mobicast* and *Location-Based Publish/Subscribe*. Finally, in a *discussion* section, we summarize the results of our review and discuss the possibility of using these abstractions and algorithms for proximity-based durable broadcast.

**Geocast.** In a geocast routing protocol, a message is disseminated to all nodes which are within a given geographic area called the *geocast region*. Thus, geocast is a type of multicast in which group membership is defined with respect to a geographic area. Various geocast routing protocols were proposed for ad hoc networks [40; 42; 8; 46; 45; 55; 54] and in particular, for vehicular ad hoc networks (VANETs) [2; 38; 75; 47; 48]. The classical geocast routing is semantically time-oblivious i.e., the geocast message is assumed to be delivered as soon as possible. In *abiding geocast* [55], the geocast message is disseminated in the geocast region for a time duration. Some papers use abiding geocast to disseminate traffic warning messages or commercial advertisement to a group of vehicles in a given zone for a given time [75; 47].

**Mobicast.** Mobicast is a class of multicast which is both location-aware and time-aware. In mobicast, a message is disseminated in an area called the *delivery zone* for a time duration  $T$ . Delivery zone can move during  $T$  and is denoted as  $Z[t]$ , where  $t$  is in  $T$ . As the delivery zone moves, some nodes enter the zone and some nodes leave the zone. The ultimate goal of mobicast is to achieve *just-in-time* message delivery, i.e.,  $Z[t]$  represents the area where the mobicast message should be delivered at time  $t$  [32]. Some mobicast protocols were proposed for wireless sensor networks [32; 11; 33] and VANETs [12]. In some of these protocols, the delivery zone is not associated to the source of the message e.g., in [32]. In [12], the delivery zone is an elliptic area around the source of the message (a moving car).

**Location-Based Publish/Subscribe.** Some authors proposed high level communication abstractions derived from the publish/subscribe paradigm, which include some type of constraint on messages in time and/or space [31; 57; 25]. In STEAM [57], messages are constrained to a location around the sender. However, there is no support for persistence and the specifications and underlying communication abstractions are not detailed. In [25], messages are persisted and can be

localized. However, they are assigned to a fixed geographical zone and do not move around with the sender. In [31], messages are persisted in a geographical zone around the publisher for a certain duration.

**Discussion.** Geocast and mobicast satisfy some but not all requirements of PBM applications for proximity-based durable broadcast. For instance, with geocast the dissemination region of a message is generally not associated with the source of the message and remains stationary, whereas the proximity-based durable broadcast requires message dissemination zone to surround the source of the message and thus move with the source of the message. Furthermore, for proximity-based durable broadcast we need abstractions which guarantee the message delivery based on the amount of time that the receiver spent in the proximity of the sender, whereas such constraints are absent from both geocast and mobicast specifications. Finally, the communication abstraction used in [31] to propagate messages around the publisher is the closest to a desirable communication abstraction for proximity-based durable broadcast, however it does not provide any guarantees.

## 1.2.2 Neighbor Detection Abstractions and Algorithms

In this section, we present a review of the main neighbor detection abstractions and algorithms in MANETs. We then discuss the possibility of using these abstractions and algorithms for proximity-based neighbor detection.

**Hello Protocols.** The majority of the existing neighbor detection algorithms in ad hoc networks belong to the *hello protocols* family [56; 71; 20; 39; 3; 36; 30]. They are based on the *basic hello protocol* first described in *Open Shortest Path First (OSPF)* routing protocol [58], which works as follows: each node in the network periodically sends *hello* messages to announce its presence to close nodes, and maintains a neighbor set. The sending frequency is denoted by  $f_{hello}$ . If a *hello* message is not received from a neighbor for a predefined amount of time, then that neighbor is discarded from the neighbor set. The problem with this approach is that if  $f_{hello}$  is too low (with respect to the speed of the nodes), then the neighbor set becomes quickly obsolete. On the other hand, if it is too high, the neighbor set remains up to date but it causes a significant waste of communication bandwidth and energy [36].

However, finding the optimal  $f_{hello}$  is not obvious and the existing solutions can only approximately solve this problem.

**Other Abstractions and Algorithms.** Although, the *hello protocols* comprise the majority of the existing neighbor detection algorithms for ad hoc networks, in the literature there exist also schemes that use different approaches than the *hello* broadcast for neighbor detection [14; 15]. For instance, in [15] Cornejo et al. define a reliable neighbor detection abstraction that establishes links over which message delivery is guaranteed. They present two region-based neighbor detection algorithms which implement the abstraction with different link establishment guarantees. The algorithms are implemented on top of a Medium Access Control (MAC) layer, which provides upper bounds on the time for message delivery. The main idea behind the first algorithm is that a node sends a *join* message some time after entering a new region to establish communication links. It also sends a *leave* message some time before leaving a region to inform the other nodes so that they can tear down their corresponding link with that node. To guarantee that these notification messages reach their destination in spite of the continuous motion of nodes, the authors define the time limits for a node to send the *join* and the *leave* messages. These time limits are obtained using the timing guarantees of the underlying MAC layer. Since a node should send a *leave* message some time before it actually leaves a region, the algorithm assumes that a node’s trajectory function is known to that node with enough anticipation to communicate with other nodes before leaving the region. The first algorithm does not guarantee the communication links when nodes are moving quickly across region boundaries. Thus, the authors introduce a second algorithm. In this new algorithm they apply a technique that overlays multiple region partitions, associating with each region partition an instance of the first algorithm. The output of each instance is then composed such that it guarantees the communication links even when nodes are moving across region boundaries. The approach applied in [15] for neighbor detection uses a relatively lower number of message broadcast compared to the *hello protocols*.

**Discussion.** All neighbor detection abstractions and algorithms discussed in this section detect the *communication neighbors* of a node, i.e., the nodes that can communicate with that node. More precisely, communication neighbors are determined by physical factors such as the transmission range of the nodes and the environment in which the algorithms are executed. However, neighbor detection in a PBM appli-



cation concerns detection of the *proximity neighbors* of a node, i.e., the nodes which are in the detection range of that node, where the detection range is a logical range defined by the semantics of the PBM application. Thus, the algorithms discussed in this section can be used for proximity-based neighbor detection if the transmission range of nodes fulfill some constraints. For instance, consider a *hello protocol* that detects the single-hop neighbors of nodes. Then, in order for this protocol to be able to detect the proximity neighbors in a PBM application, the transmission range of nodes should be greater than or equal to the detection range of the PBM application.

As already discussed, the choice of  $f_{hello}$  has a direct influence on detection effectiveness and energy efficiency of a *hello protocol*. Thus, if a *hello protocol* is used to implement a PBM application, its  $f_{hello}$  should be carefully chosen in order to respect the detection effectiveness as well as the energy *efficiency* requirements of proximity-based neighbor detection.

Finally, note that all neighbor detection abstractions and algorithms discussed in this section, provide only the set of current neighbors and not the past or future neighbors as it is specified in the requirements of proximity-based neighbor detection.

### 1.3 Problem Statement and Research Questions

The overall question that we address in this thesis is the following: *which abstractions and algorithms can be used to build PBM applications in MANETs?* In Section 1.1.1, we defined *proximity-based durable broadcast* and *proximity-based neighbor detection* as the two main requirements of PBM applications. Thereby, we split the aforementioned question into two main research questions **Q1** and **Q2** where each question is related to one of the requirements. In the following, we present each research question and then give the intuition behind the way we address it in the thesis.

**Q1** *Which abstractions and algorithms can be used for proximity-based durable broadcast?*

As already discussed in Section 1.2, the existing communication abstractions and algorithms for MANETs cannot entirely fulfill the requirements of PBM applications for proximity-based durable broadcast. Thus, our goal is to design and implement a new abstraction for proximity-based durable broadcast. Given the broadcast message, the broadcast range and the broadcast duration, our abstraction should enable

a node to broadcast the message in the defined range (around the sender) for the given time duration. As already described in Section 1.1.1, the message delivery guarantees of the proximity-based durable broadcast are not the same in all PBM applications. Accordingly, the abstraction that we design for the proximity-based durable broadcast can have more than one variant, such that the variants differ in their message delivery guarantees.

Intuitively, our proximity-based durable broadcast service can be built using an underlying proximity-based broadcast abstraction. This leads to the following sub-question:

**Q1.1** *Which abstractions and algorithms can be used for proximity-based broadcast?*

Intuitively, a proximity-based broadcast abstraction should be such that given the broadcast message and the broadcast range, it enables a node to broadcast the message in the defined range. Moreover, it should be implementable in both single-hop and multi-hop cases. Finally, it can have different variants based on message delivery guarantees required by the upper proximity-based durable broadcast layer.

**Q2** *Which abstractions and algorithms can be used for proximity-based neighbor detection?*

We consider two approaches to address this question: (1) using the *hello protocols* for proximity-based neighbor detection; (2) designing and implementing a new neighbor detection abstraction. In the following, we discuss the two approaches and the research questions related to each approach.

**(1) Using the hello protocols.** Here, for simplicity's sake we only consider the *hello protocols* that detect the single-hop neighbors of a node. As already discussed in Section 1.2, such *hello protocols* can be used to detect current neighbors of nodes in a PBM application if the transmission range of nodes is greater than or equal to the detection range of the PBM application. According to the characteristics of proximity-based neighbor detection (defined in Section 1.1.1) we know that the *hello protocols* are not the ideal neighbor detection algorithms for PBM applications since they only detect the current neighbors. However, these protocols constitute the majority of existing neighbor detection algorithms for MANETs and are easy to implement (especially in the single-hop case). Thus, if one can adjust the input param-

eters of the *hello protocols* so that they satisfy other requirements of proximity-based neighbor detection such as *effectiveness* and *efficiency*, then these protocols can be used to build those PBM applications that need only current neighbor detection. This leads us to the following subquestion:

**Q2.1** *How can hello protocols be used for effective and efficient proximity-based neighbor detection?*

Intuitively, *the effectiveness* of neighbor detection can be increased by increasing the transmission power and decreasing the time interval between two consecutive broadcasts of the *hello message*. However, these methods can also increase the energy consumption of nodes and thus, decrease *the efficiency* of neighbor detection. Therefore, we should find a tradeoff between *effectiveness* and *efficiency*.

**(2) Designing and implementing a new neighbor detection abstraction.**

Our goal is to design and implement a new abstraction that matches the characteristics of proximity-based neighbor detection described in Section 1.1.1. In particular, it should enable a node to detect not only its current neighbors, but also its past and future neighbors (up to some bounded time interval). Similar to a failure detector, our neighbor detection abstraction should have a *completeness* property and an *accuracy* property. The *completeness* property guarantees to detect all past, present and future neighbors (up to some bounded time interval). This property in fact guarantees the *effectiveness* of neighbor detection (defined in Section 1.1.1) required by PBM applications. The *accuracy* property basically guarantees that no false detection occurs. This leads us to the following subquestion:

**Q2.2** *How can our new abstraction for proximity-based neighbor detection be implemented?*

To build such a neighbor detector service, our intuition is to equip each node with a mobility predictor service that can accurately predict the future locations of that node up to some bounded time interval  $\Delta_{predict}$  in the future. We also use a moving entity that travels through the network and collects the location predictions of nodes. Then, there exist two methods: either the moving entity computes the list of current and future neighbors of each node based on the location predictions and distributes the lists to nodes or the moving entity directly distributes the location predictions to nodes so that they can find their current and future neighbors based on the distributed location predictions. Each node also stores the distributed information

(the neighbor list or the location predictions, depending on the used method), so that later it will be able to be queried about its past neighbors. This leads us to the six following subquestions:

**Q2.2.1** *Which abstractions and algorithms can be used for the mobility predictor?*

Our goal is to design a mobility predictor abstraction that guarantees accurate location prediction up to  $\Delta_{predict}$ . Since in PBM applications mobile devices are held by people, our intuition is that the mobility predictor abstraction can be probabilistically implemented by one of the existing human mobility prediction algorithms [68; 64].

**Q2.2.2** *Which abstraction and algorithm can be used for the moving entity?*

In the literature of MANETs, there exist different types of moving entities such as ferry nodes [76; 78; 77; 6], Data MULEs [67] and virtual mobile nodes [19], which are used for different tasks such as routing, data collection or group communication. The most important fact for choosing the proper moving entity is that its movement should be programmable, however, it should not disrupt the normal movement of the nodes. The reason is that in PBM applications the wireless devices are used by ordinary people who are not amenable to following instructions as to where their devices may travel.

**Q2.2.3** *What is the minimum value of  $\Delta_{predict}$  for which our new abstraction for proximity-based neighbor detection can be correctly implemented?*

In order for our neighbor detector service to be correctly implemented,  $\Delta_{predict}$  of the mobility predictor service should be long enough so that the location predictions or neighbor lists do not expire before they are distributed by the moving entity. Intuitively, this value of  $\Delta_{predict}$  can be defined as a function of the time that the moving entity spends to travel across the network.

**Q2.2.4** *Can the use of multiple moving entities improve the neighbor detection?*

Our intuition is that if more than one moving entity is used to implement our neighbor detector service, then each moving entity can be responsible for collection and distribution in one part of the network. This reduces the traveling time of each moving entity, which in turn reduces  $\Delta_{predict}$  required for the correct neighbor detection.

**Q2.2.5** *Can neighbor detection be performed despite the failure of moving entities?*

Regardless of its nature (ferry node, Data MULE or virtual mobile node), a moving entity can crash and lose all its stored information. This crash can be critical for the neighbor detection process. Thus, one of the issues that should be investigated is how the failure of the moving entities can be tolerated by our neighbor detector algorithm.

**Q2.2.6** *How can the implementation of our new abstraction for proximity-based neighbor detection be efficient?*

One of the requirements of the proximity-based neighbor detection is *efficiency* of neighbor detection in terms of energy consumption (see Section 1.1.1). Therefore, the implementation of our new neighbor detector abstraction should also be energy efficient. Intuitively, the energy efficiency of our implementation depends on the nature of the moving entity as well as the number of messages exchanged between the moving entity and the nodes.

## 1.4 Thesis Overview

The thesis is written in the *thesis by publication* format. More precisely, each core chapter of the thesis corresponds to at least one article which is already published in a highly-visible international conference or journal. The complete list of publications related to this thesis can be found at the end of this chapter (see Section 1.5). We also indicate the list of publications related to each chapter at the beginning of that chapter. In particular, Chapter 3 corresponds to two articles: one conference article and one journal article. However, since the journal article is the extended version of the conference article, we only include the content of the journal article for this chapter. The same principle is applied for Chapter 5 i.e., it only contains the content of the journal article. The advantage of using the *thesis by publication* format is that each chapter of the thesis can be read independently of the other chapters. Note however that this format may result in some content redundancies.

The thesis is divided into two main parts where each part addresses a research question and its subquestions stated in Section 1.3. More precisely, the first part of the thesis focuses on introducing abstractions and algorithms that can be used for

proximity-based durable broadcast. It addresses research question **Q1** and its subquestion. The second part focuses on abstractions and algorithms that can be used for proximity-based neighbor detection. Thus, it addresses research question **Q2** and its subquestions. Note that the parts of the thesis are presented in the chronological order i.e., the order in which they were completed during this research. The two parts are related in the sense that some of abstractions and algorithms introduced in the first part can be implemented using the abstractions and algorithms introduced in the second part and vice-versa. For instance, the spotcast abstraction (introduced in Chapter 2) can be implemented using the time-limited neighbor detector abstraction (introduced in Chapter 4)<sup>1</sup> and the time-limited neighbor detector abstraction (introduced in Chapter 4) can be implemented using the scoped-broadcast abstraction (introduced in Chapter 2). Below, we take a closer look at each part of the thesis and its chapters.

## Part I

This part introduces abstractions and algorithms that can be used for proximity-based durable broadcast. It addresses research question **Q1** and its subquestion. It includes the following chapter:

**Chapter 2.** In this chapter, we address **Q1** by introducing *spotcast*, which is a new communication abstraction for proximity-based durable broadcast in MANETs. Spotcast enables a node to disseminate a message for a given time duration to all nodes located within a given range. Spotcast has three variants which differ in their timing guarantees for message delivery. These guarantees are defined based on different requirements of PBM applications. We also address **Q1.1** by introducing *scoped broadcast* abstraction, which enables a node to send a message to all nodes located within a given range. Scoped broadcast has two variants: a synchronous variant and asynchronous variant. We present a *generic* algorithm that can implement the three spotcast variants using different scoped broadcast variants and different types of message buffers. We present a proof of correctness for the algorithm. We also dis-

---

<sup>1</sup> In Chapter 2, we only present a simple and generic algorithm that implements the spotcast variants using different scoped broadcast variants. However, to implement the spotcast variants, one can also think of using the time-limited neighbor detector abstraction introduced in Chapter 4, which may result in a more resource efficient implementation.

cuss how scoped broadcast variants can be implemented in single-hop and multi-hop cases.

## Part II

This part presents abstractions and algorithms that can be used for proximity-based neighbor detection. It addresses research question **Q2** and its subquestions. It includes the following chapters:

**Chapter 3.** In this chapter, we address **Q2** and in particular **Q2.1** by trying to adjust the parameters of *hello protocols* for effective and efficient proximity-based neighbor detection. In order to do so, we perform a simulation-based study using ns-2 network simulator. More precisely, we assume that nodes communicate using the *IEEE 802.11a* technology [34] and we define a *strategy* as a pair of the transmission power and the time interval between two consecutive broadcasts of the *hello* message. Then, we consider a set of strategies defined based on the characteristics of current smartphones. Moreover, since the quality of radio signals (and consequently the effectiveness) is affected by the environment attenuation, for our study we consider three typical urban environments where PBM applications are used i.e., *indoor with hard partitions* (corresponding to offices with thick walls), *indoor with soft partitions* (corresponding to indoor exhibitions with temporary partitions) and *outdoor urban areas* (corresponding to a music festival in downtown). To simulate these environments, we use a radio propagation model known for modeling the obstructed urban environments called *Log-Normal Shadowing* (LNS). Then, we evaluate the effectiveness and efficiency of the set of strategies in each environment. We deduce that the efficiency is independent of the environment. We also discuss the impact of changing transmission power and broadcast interval on effectiveness and efficiency. We identify the most effective strategy and the most efficient strategy in each environment. We observe that we cannot find a strategy that maximizes both effectiveness and efficiency in any environment. The reason is that, regardless of environment, effectiveness and efficiency are in conflict with each other. We then propose an approach to make a tradeoff between effectiveness and efficiency. Accordingly, we find the tradeoff strategy in each environment and we show that it has a relatively good effectiveness and efficiency compared to other strategies.

**Chapter 4.** In this chapter, we address **Q2** by introducing a new abstraction called the *time-limited neighbor detector* for proximity-based neighbor detection. It enables a node to detect its neighbors in the past, present and up to some bounded time interval in the future. It has a *completeness* property that guarantees to detect all past, present and future neighbors (up to some bounded time interval). It has also an *accuracy* property which guarantees that no false detection occurs. We address **Q2.2** and some of its subquestions by presenting an algorithm that implements the *time-limited neighbor detector*. We assume that each node has access to a mobility predictor service that accurately predicts its locations up to some bounded time interval  $\Delta_{predict}$  in the future. Thus, we address **Q2.2.1** by introducing an abstraction for the mobility predictor service, we also give some hints on its implementability (we give a more detailed discussion regarding the implementability of this abstraction in Chapter 5). Our algorithm uses a single virtual mobile node that travels through the network, collects the location predictions of nodes and distributes the neighbor lists, which it creates based on the collected location predictions, to nodes. Thus, we address **Q2.2.2** by using a virtual mobile node as the moving entity. We also present an algorithm (first introduced in [19]) that implements the virtual mobile node abstraction using a set of real nodes based on a replicated state machine approach. The reason behind our choice of a virtual mobile node as the moving entity is that among all the existing moving entities in the literature, virtual mobile node is the only moving entity that does not disrupt the node mobility and at the same time its movement can be predefined. We prove the correctness of the neighbor detector algorithm under certain conditions. In particular, the algorithm is correct if the virtual mobile node is correct i.e., it never fails. As already described, in this chapter we address two subquestions of **Q2.2** namely, **Q2.2.1** and **Q2.2.2**. The other subquestions of **Q2.2** are addressed in the next chapter that is, Chapter 5.

**Chapter 5.** In this chapter, we address **Q2** and in particular **Q2.2** by introducing an algorithm that implements the *time-limited neighbor detector* abstraction (defined in Chapter 4) using  $n = 2^k$  virtual mobile nodes, where  $k$  is a non-negative integer. The algorithm implements the neighbor detector for nodes located in a circular region. We also assume that each node has access to the mobility predictor service defined in Chapter 4. The key idea of the algorithm is that the virtual mobile nodes regularly collect location predictions of nodes from different subregions, meet to share what they have collected with each other and then distribute the collected location predictions to nodes. Thus, each node can find its neighbors at current



and future times based on the distributed location predictions. It can also store the location predictions so it can be queried about its past neighbors. We show that our algorithm is correct in periodically well-populated regions. Compared to the algorithm proposed in Chapter 4, this algorithm has two advantages: (1) it tolerates the failure of one to all virtual mobile nodes; (2) as  $n$  grows, it correctly implements the neighbor detector with smaller values of  $\Delta_{predict}$ . Intuitively, this is because as  $n$  grows, the circular region is divided into more and consequently smaller subregions and each virtual mobile node spends less time to travel through its subregion. This feature makes the real-world deployment of the neighbor detector easier since with the existing prediction methods, location predictions usually tend to become less accurate as  $\Delta_{predict}$  increases. As already described, in Chapter 4 we only give some hints on how the mobility predictor service can be implemented. Thus, in this chapter we address **Q2.2.1** by discussing in detail how the mobility predictor service can be implemented by one of the existing human mobility prediction methods. We also address **Q2.2.3** by defining the minimum value of  $\Delta_{predict}$  for which the algorithm is correct. We address **Q2.2.4** by showing that increasing the number of virtual mobile nodes can decrease the minimum value of  $\Delta_{predict}$  required for the correctness of the algorithm. We address **Q2.2.5** by proving that the algorithm is correct despite of failure of one to all virtual mobile nodes. Finally, we address **Q2.2.6** by showing that the cost of the algorithm (in terms of communication) scales linearly with the number of virtual mobile nodes. In fact, communication is the main cause of energy consumption in a network executing the algorithm. Thus, communication cost influences the efficiency of the algorithm in terms of energy. We also propose a set of optimizations which reduce the communication cost of the algorithm.

## 1.5 List of Publications

This thesis is based on the publications listed below.

- B. Bostanipour, B. Garbinato and A. Holzer, Spotcast – A communication abstraction for proximity-based mobile applications, *In Proceedings of The 11th IEEE International Symposium on Network Computing and Applications (IEEE NCA '12)*, pp. 121–129, Cambridge, Massachusetts, USA, IEEE, 2012.
- B. Bostanipour and B. Garbinato, Improving neighbor detection for proximity-based mobile applications, *In Proceedings of The 12th IEEE International Sym-*

*posium on Network Computing and Applications (IEEE NCA'13)*, pp. 177–182, Cambridge, Massachusetts, USA, IEEE, 2013.

- B. Bostanipour and B. Garbinato, Using virtual mobile nodes for neighbor detection in proximity-based mobile applications, *In Proceedings of The 13th IEEE International Symposium on Network Computing and Applications (IEEE NCA'14)*, pp. 9–16, Cambridge, Massachusetts, USA, IEEE, 2014.
- B. Bostanipour and B. Garbinato, Effective and efficient neighbor detection for proximity-based mobile applications, *In Elsevier Computer Networks Journal*, vol. 79, pp. 216–235, 2015.
- B. Bostanipour and B. Garbinato, Neighbor detection based on multiple virtual mobile nodes, *In Proceedings of the 24th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'16)*, pp. 322–327, Heraklion, Greece, IEEE, 2016.
- B. Bostanipour and B. Garbinato, A neighbor detection algorithm based on multiple virtual mobile nodes for mobile ad hoc networks, *currently under minor revision for publication in Elsevier Computer Networks Journal*, 2016.

## References

- [1] Abolhasan, M., Wysocki, T., Dutkiewicz, E.: A Review of Routing Protocols for Mobile Ad Hoc Networks, *Ad Hoc Networks*, 2(1):1–22, (2004).
- [2] Bachir, A., Benslimane, A.: A Multicast Protocol in Ad hoc Networks Inter-vehicle Geocast. In: *Proceedings of IEEE Semiannual Vehicular Technology Conference, VTC'03*, 4:2456–2460, IEEE, (2003).
- [3] Bakht, M., Trower, M., Kravets, R., Searchlight: Helping Mobile Devices Find their Neighbors. In *ACM SIGOPS Oper. Syst.*, vol. 45, no. 3, pp. 71–76, (2012).
- [4] Basagni, S., Chlamtac, I., Syrotiuk, V.-R., Woodward, B.-A.: A Distance Routing Effect Algorithm for Mobility (DREAM). In: *Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom '98*, pp. 76–84, (1998).
- [5] Ben-Or, M.: Another Advantage of Free Choice (extended abstract): Completely Asynchronous Agreement Protocols. In: *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, PODC '83*, pp. 27–30, (1983).

- [6] Bin Tariq, M. M., Ammar, M., Zegura, E., Message Ferry Route Design for Sparse Ad Hoc Networks with Mobile Nodes, In *Proc. ACM MobiHoc'06*, pp. 37–48, (2006).
- [7] Bizzabo. <https://www.bizzabo.com/>
- [8] Boleng, J., Camp, T., Tolety, V.: Mesh-Based Geocast Routing Protocols in an Ad hoc Network. In: Proceedings of 15th International Parallel and Distributed Processing Symposium, IPDPS '01, pp. 1924–1933, IEEE Comput. Soc., (2001).
- [9] Borran, F., Round-Based Consensus Algorithms, Predicate Implementations and Quantitative Analysis, EPFL Thesis, no 4839, (2011).
- [10] Chandra, T. D., Toueg, S.: Unreliable Failure Detectors for Reliable Distributed Systems. *JACM.*, 43:225–267, (1996).
- [11] Chen, Y.S., Ann, S.Y., Lin, Y.W.: VE-Mobicast: A Variant-Egg-Based Mobicast Routing Protocol for SensorNet. *ACM Wireless Networks.*, 14:199–218, (2008).
- [12] Chen, Y.S., Lin, Y.W., Lee, S.L.: A Mobicast Routing Protocol in Vehicular Ad-hoc Networks. *Mob. Netw. Appl.*, 15:20–35, (2010).
- [13] Chockler, G., Demirbas, M., Gilbert, S., Newport, C., Nolte, T.: Consensus and Collision Detectors in Wireless Ad hoc Networks. In: Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, PODC'05, pp. 197–206, ACM, (2005).
- [14] Cornejo, A., Viqar, S., Welch, J. L., Reliable Neighbor Discovery for Mobile Ad Hoc Networks, In *Proc. DIALM-PODC'10*, pp. 63-72, (2010).
- [15] Cornejo, A., Viqar, S., Welch, J. L., Reliable Neighbor Discovery for Mobile Ad Hoc Networks, In *Ad Hoc Networks*. 12 (January 2014), pp. 259–277, (2014).
- [16] Defago, X., Schiper, A., Urban, P.: Total Order Broadcast and Multicast Algorithms: Taxonomy and survey. *ACM COMPUTING SURVEYS*, 36:2004, (2004).
- [17] Dolev, D., Dwork, C., Stockmeyer, L.: On the Minimal Synchronism needed for Distributed Consensus. *JACM*, 34:77–97, (1987).
- [18] Dolev, D., Friedman, R., Keidar, I., Malkhi, D.: Failure Detectors in Omission Failure Environments. Technical report, Department of Computer Science, Cornell University, (1997).
- [19] Dolev, S., Gilbert, S., Lynch, N. A., Schiller, E., Shvartsman, A. A., Welch, J. L., Virtual Mobile Nodes for Mobile Ad Hoc Networks, In *Proc. DISC'04*, pp. 230–244, (2004).
- [20] Dutta, P., Culler, D., Practical Asynchronous Neighbor Discovery and Rendezvous for Mobile Sensing Applications. In *Proc. ACM SenSys'08*, (2008).

- [21] Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the Presence of Partial Synchrony. *JACM.*, 35:288–323, (1988).
- [22] Ezhilchelvan, P., Mostefaoui, A., Raynal, M.: Randomized Multivalued Consensus. In: Proceedings of the 4th International Symposium on Object-Oriented Real-Time Computing, pp. 195–200. IEEE Computer Society Press, (2001).
- [23] FireChat. <https://itunes.apple.com/en/app/firechat/id719829352?mt=8>.
- [24] Fischer, M., Lynch, N., Paterson, M.: Impossibility of Distributed Consensus with one Faulty Process. *JACM*, 32:374–382, (1985).
- [25] Frey, D., Roman, G.: Context-Aware Publish Subscribe in Mobile Ad hoc Networks. In: Proceedings of the 9th International Conference on Coordination Models and Languages, Coordination’07, pp. 37–55, Springer, (2007).
- [26] Friedman, R., Mostefaoui, A., Raynal, M.: On the Respective Power of  $\diamond_p$  and  $\diamond_s$  to Solve One-shot Agreement Problems. *IEEE Trans. Parallel Distrib. Syst.*, 18:589–597, (2007).
- [27] Garbinato, B., Holzer, A., Vessaz, F.: Context-Aware Broadcasting Approaches in Mobile Ad Hoc Networks. *Comput. Netw.*, 54:1210–1228, (2010).
- [28] Hadzilacos, V., Toueg, S.: Fault-Tolerant Broadcasts and Related Problems. pp. 97–145, ACM Press/Addison-Wesley Publishing Co., New York, (1993).
- [29] Haas, Z. J., Pearlman, M. R.: The Zone Routing Protocol (ZRP) for Ad Hoc Networks. *IETF MANET, Internet Draft*, (1997).
- [30] He, D., Mitton, N., Simplot-Ryl, D., An Energy Efficient Adaptive HELLO Algorithm for Mobile Ad Hoc Networks, In *Proc. ACM MSWiM’13*, pp. 65–72, (2013).
- [31] Holzer, A., Eugster, P., Garbinato, B.: Evaluating Implementation Strategies for Location-based Multicast Addressing. *IEEE TMC*, (2013).
- [32] Huang, Q., Lu, C., Roman, G.: Mobicast : Just-in-Time Multicast for Sensor Networks under Spatiotemporal Constraints. In: Proceedings of the International Conference on Information Processing in Sensor Networks, IPSN’03, pp. 442–457, (2003).
- [33] Huang, Q., Lu, C., Roman, G.: Reliable Mobicast via Face-Aware Routing. In: Proceedings of IEEE International Conference on Computer Communications, INFOCOM’04, pp. 205–217, (2004).
- [34] IEEE 802.11: Wireless LAN MAC and Physical Layer Specifications. June (1999).

- [35] iGroups: Apples New iPhone Social app in development. Patently Apple. <http://tinyurl.com/y9cwvmx>
- [36] Ingelrest, F., Mitton, N., Simplot-Ryl, D., A Turnover Based Adaptive Hello Protocol For Mobile Ad Hoc and Sensor Networks, In *Proc. MASCOTS'07*, pp. 9–14, (2007).
- [37] Johnson, D., Maltz, D. A.: Dynamic Source Routing in Ad hoc Wireless Networks. Mobile Computing (Imielinski, T., Korth, H., eds.), pp. 153–181, Kluwer Acad. Publ., (1996).
- [38] Joshi, H.P., Sichitiu, M., Kihl, M.: Distributed Robust Geocast Multicast Routing for Inter-vehicle Communication. In: Proceedings of Weird Workshop on WiMAX, Wireless and Mobility, Weird'07, pp. 9–21, (2007).
- [39] Kandhalu, A., Lakshmanan, K., Rajkumar, R. R., U-Connect: a Low-Latency Energy-Efficient Asynchronous Neighbor Discovery Protocol, In *IPSN'10*, pp. 350–361, (2010).
- [40] Ko, Y.B., Vaidya, N.H.: Geocasting in Mobile Ad hoc Networks: Location-Based Multicast Algorithms. In: Proceedings of Second IEEE Workshop on Mobile Computing Systems and Applications, WMCSA'99, pp. 101–110, IEEE, (1999).
- [41] Ko, Y.B., Vaidya, N.H.: Location-aided Routing (LAR) in Mobile Ad hoc Networks. *Wirel. Netw.*, 6:307–321, (2000).
- [42] Ko, Y.B., Vaidya, N.H.: Geotora: a Protocol for Geocasting in Mobile Ad hoc Networks. In: Proceedings of International Conference on Network Protocols, ICNP'00, (00-010):240–250, IEEE Comput. Soc., (2000).
- [43] Lamport, L.: Ti clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21:558–565, (1978).
- [44] Lamport, L.: Using Time instead of Timeout for Fault-tolerant Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 6:254–280, (1984).
- [45] Lee, S.H., Ko, Y.B.: Geometry-Driven Scheme for Geocast Routing in Mobile Ad Hoc Networks. In: Proceedings of the IEEE Vehicular Technology Conference, VTC'06, pp. 638–642, IEEE, (2006).
- [46] Liao, W. H., Tseng, Y.C., Lo, K.L. , Sheu, J. P.: GeoGRID: A Geocasting Protocol for Mobile Ad Hoc Networks Based on GRID. *JIT.*, 1–2:23–32, (2000).
- [47] Lim, Y., Ahn, S., Cho, K.H.: Abiding Geocast for Commercial Ad Dissemination in the Vehicular Ad hoc Network. In: Proceedings of IEEE International

- Conference on Consumer Electronics, ICCE'11, pp. 115–116, (2011).
- [48] Lin, Y., Chen, Y., Lee, S.: Routing Protocols in Vehicular Ad hoc Networks: A Survey and Future Perspectives. *J. Inf. Sci. Eng.*, 26(3):913–932, (2010).
- [49] Liu, C., Kaiser, J.: A Survey of Mobile Ad Hoc network Routing Protocols. Technical Report 2003-08, University of Ulm, (2003).
- [50] Local Multiplayer Apps. <http://appcrawlr.com/ios-apps/best-apps-local-multiplayer>
- [51] LocoPing. <http://www.locoping.com/>
- [52] LoKast. <http://www.lokast.com/>.
- [53] Lou, W., Wu, J.: Double-Covered Broadcast (DCB): A simple Reliable Broadcast Algorithm In MANETs. In: Proceedings of IEEE International Conference on Computer Communications, INFOCOM'12, (2004).
- [54] Maihöfer, C.: A Survey of Geocast Routing Protocols. *IEEE Communications Surveys and Tutorials*, 6(1-4):32–42, (2004).
- [55] Maihöfer, C., Leinmüller, T., Schoch, E.: Abiding Geocast: Time-Stable Geocast for Ad hoc Networks. In: Proceedings of the 2nd ACM International Workshop on Vehicular Ad hoc Networks, VANET '05, pp. 20–29, ACM, (2005).
- [56] McGlynn, M. J., Borbash, S. A., Birthday Protocols for Low Energy Deployment and Flexible Neighbor Discovery in Ad Hoc Wireless Networks. In *Proc. ACM MobiHoc'01*, pp. 137–145, (2001).
- [57] Meier, R., Cahill, V.: On Event-Based Middleware for Location-Aware Mobile Applications. *IEEE TSE*, 36(3):409–430, (2010).
- [58] Moy, J., OSPF – Open Shortest Path First, RFC 1583, (1994).
- [59] Murthy, S., Garcia-Luna-Aceves, J.: An Efficient Routing Protocol for Wireless Networks. *ACM Mobile Networks and App. J.*, Special Issue on Routing in Mobile Communication Networks, pp. 183-97, (1996).
- [60] Nikaiein, N., Bonnet, C., Nikaiein, N.: Harp - Hybrid Ad-Hoc Routing Protocol. In: Proceedings of International Symposium on Telecommunications, IST'01, (2001).
- [61] Oliveira, R.: Solving Consensus : from Fair-lossy Channels to Crash-recovery of Processes.. PhD thesis, EPFL, (2000).
- [62] Perkins, C.E., Bhagwat, P.: Highly dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In: Proceedings of ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'94), vol. 24, no.4, pp.234-244, (1994)

- [63] Perkins, C.E., Royer, E.M.: Ad-hoc On-demand Distance Vector Routing. In: Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, WMCSA'99, pp. 90–100, (1999).
- [64] Scellato, S., Musolesi, M., Mascolo, C., Latora, V., Campbell, A. T., Nextplace: a Spatio-Temporal Prediction Framework for Pervasive Systems, In *Proc. Pervasive'11*, pp. 152–169, (2011).
- [65] Schneider, F.-B.: Implementing Fault-tolerant Services using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22:299–319, (1990).
- [66] Schneider, Fred B.: Replication Management Using the State-machine Approach. ACM Press/Addison-Wesley Publishing Co., pp. 69–197, (1993).
- [67] Shah, R., Roy, S., Jain, S., Brunette, W., Data MULEs: Modeling a Three-Tier Architecture for Sparse Sensor Networks, In *Elsevier Ad Hoc Networks Journal*, vol. 1, pp. 215–233, (2003).
- [68] Song, L., Deshpande, U., Kozat, U.C., Kotz, D., Jain, R., Predictability of WLAN Mobility and its Effects on Bandwidth Provisioning. In *Proc. IEEE INFOCOM'06*, (2006).
- [69] Tanenbaum, A.S.: Computer Networks, Computer Science, Prentice Hall PTR, (2003).
- [70] Vessaz, F., System-Level Support for Mobile Ad hoc Communications : an Algorithmic and Practical Approach, UNIL Thesis, (2013).
- [71] Wattenhofer, G., Alonso, G., Kranakis, E., Widmayer, P., Randomized Protocols for Node Discovery in Ad Hoc, Single Broadcast Channel Networks, In *Proc. IPDPS'03*, (2003).
- [72] Waze. <https://www.waze.com/en/>
- [73] WhosHere. <http://whoshere.net/>
- [74] Wu, W., Cao, J., Yang, J., Raynal, M.: Design and Performance Evaluation of Efficient Consensus Protocols for Mobile Ad hoc Networks. *IEEE Trans. Computers*, (2007).
- [75] Yu, Q., Heijenk, G.: Abiding Geocast for Warning Message Dissemination in Vehicular Ad hoc Networks. In: Proceedings of International Conference On Communications, ICC'08, pp. 400–404, (2008).
- [76] Zhao, W., Ammar, M., Message Ferrying: Proactive Routing in Highly-Partitioned Wireless Ad Hoc Networks, In *Proc. IEEE FTDCS'03*, pp. 308–314, (2003).

- [77] Zhao, W., Ammar, M., Zegura, E., A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks, In *Proc. ACM MobiHoc'04*, pp. 187–198, (2004).
- [78] Zhao, W., Ammar, M., Zegura, E., Controlling the Mobility of Multiple Data Transport Ferries in a Delay-Tolerant Network, In *Proc. IEEE INFOCOM'05*, vol. 2, pp. 1407–1418, (2005).



Part I

Abstractions and Algorithms for  
Proximity-Based Durable Broadcast



## Chapter 2

# Spotcast – A Communication Abstraction for Proximity-Based Mobile Applications

**Abstract** We introduce *spotcast*, a new communication abstraction specifically aimed at the development of mobile proximity-based applications running in mobile ad hoc networks (MANETs). Our motivation lies in the fact that traditional communication abstractions, typically broadcast primitives with strong consistency guarantees, do not adequately capture the intrinsic *here-and-now* nature of such applications. Rather, developers need a communication abstraction offering the notion of *proximity-based diffusion* and some level of *message durability*, which is precisely what spotcast provides. We illustrate how spotcast can be used to implement mobile applications and we shortly discuss the correctness and the implementability of the spotcast abstraction in MANETs.

### Publication:

B. Bostanipour, B. Garbinato and A. Holzer, Spotcast – A communication abstraction for proximity-based mobile applications, *In Proceedings of The 11th IEEE International Symposium on Network Computing and Applications (IEEE NCA'12)*, pp. 121–129, Cambridge, Massachusetts, USA, IEEE, 2012.

## 2.1 Distributed Systems: Evolution or Revolution?

The importance of distributed systems has grown dramatically in the past 20 years, due to the exponential increase in the number of computers of all sizes interconnected via the Internet. Driven by this growth, distributed applications broke free from intranets where they tended to be strictly confined until the mid-nineties. As a natural consequence, they became ubiquitous, as testified by today's omnipresence of

communication-oriented applications in our daily lives, typically running on mobile devices such as smartphones.

### 2.1.1 First Evolution

Examples of this evolution are the strong interests for peer-to-peer file sharing in the late nineties and the growing interest for cloud computing today. To address the evolving challenges posed by such open and potentially large-scale distributed systems, the research community has been constantly adapting the same communication abstractions it originally devised for intranet-oriented applications (database applications,  $n$ -tier services, etc). Such communication abstractions typically include various flavors of reliable broadcast and multicast [14], as well as different variants of consensus [23; 10; 5].

### 2.1.2 Then Paradigm Shift

When looking at how computer systems are being used today, particularly in the *mobile arena*, it is becoming more and more obvious that the evolution of distributed system is now reaching a breaking point. That is, deep changes are no longer occurring along the line of *how we do things* in distributed systems (client/server, peer-to-peer, cloud computing, etc.) but rather along the line of *what we do* with distributed systems (mobile online gaming, location-based services, mobile multimedia and interactive entertainment, etc.).

A good example of this ongoing paradigm shift can be found in the exploding number of social networking applications available on mobile devices: almost 10,000 such applications can be found in Apple's AppStore today,<sup>1</sup> and probably as many in the corresponding Android market. When put in perspective, mobile social networking is a particular case of an even larger set of mobile distributed applications known as *proximity-based mobile applications*. Contrary to typical distributed applications that flourished in the past two decades, this new blend of mobile applications tend to exhibit an intrinsic and somehow elastic *here-and-now* nature, as illustrated hereafter with concrete examples.

---

<sup>1</sup> March 2012.

### 2.1.2.1 On-the-spot Survey

The notion of *on-the-spot survey*, proposed by mobile applications such as SpotMe ([www.spotme.com](http://www.spotme.com)), iSurvey ([www.isurveysoft.com](http://www.isurveysoft.com)), or Voxco Mobile ([www.voxco.com](http://www.voxco.com)), is a good example of mobile proximity-based applications. Here the idea is for a user to ask questions to other nearby users, e.g., a teacher surveying students in the classroom, a speaker surveying customers at a marketing event, etc. This application requires a rather strict *here-and-now* semantics, since it is important for the surveyor to get feedback from the audience in a timely manner.

### 2.1.2.2 Social Radar

The concept of *social radar*, proposed by various mobile applications such as FourSquare ([www.foursquare.com](http://www.foursquare.com)), FriendThem ([www.friendthem.com](http://www.friendthem.com)) or Blendr ([www.blendr.com](http://www.blendr.com)), is another example of proximity-based mobile application. Intuitively, a social radar provides mobile device users with the ability to spot other users around them. This application can live with a somehow more elastic *here-and-now* semantics, since it is sufficient that users staying close enough for long enough eventually detect each other.

### 2.1.2.3 Mobile Photo Sharing

The idea behind *mobile photo sharing*, proposed by applications such as Instagram ([www.instagram.com](http://www.instagram.com)), Streamzoo ([www.streamzoo.com](http://www.streamzoo.com)), or Picplz ([www.picplz.com](http://www.picplz.com)), is to allow users participating in some social event to take photos with their mobile devices and to share them with other users present at that event. This application exhibits a rather loose *here-and-now* semantics: even if it might not be possible for two users to share all their photos while at the event, e.g., because one has to leave earlier, as soon as they get together again (possibly long after the event), the sharing can resume.

### 2.1.3 Spotcast: A New Communication Abstraction

Traditional communication abstractions, such as atomic broadcast or consensus, were devised long before mobile proximity-based applications became ubiquitous. For this reason, we advocate that this new blend of distributed applications demands a new communication paradigm, i.e., one that appropriately captures their here-and-now nature. To address this need, this paper introduces *spotcast*, a new communication abstraction that enables a mobile entity to disseminate a message in a defined range around it (here) for a defined duration (now). To be more precise, we actually present three variants of the spotcast abstraction, with slightly different delivery guarantees, in order to support various communication semantics required by mobile proximity-based applications.

### 2.1.4 Roadmap

This paper is organized as follows. In Section 2.2, we describe our system model. In Section 2.3, we introduce our novel spotcast communication abstraction. In Section 2.4, we present the implementation of spotcast, together with a discussion on its implementability. In Section 2.5, we discuss the implementability of the underlying scoped broadcast service. Finally, we discuss related work in Section 2.6 before concluding in Section 2.7 with a perspective on future work.

## 2.2 System Model

We consider a mobile ad-hoc network (MANET) consisting of a finite set of  $n$  processes  $P = \{p_1, \dots, p_n\}$ . We use the terms *process* and *node* interchangeably. Processes are in a two-dimensional plane. Each process has a unique identifier. Processes are *mobile* i.e., their geographic location can change unpredictably over time. Processes can experience *crash* failures. A crash faulty process stops prematurely. Prior to stopping, it behaves correctly.<sup>2</sup>

---

<sup>2</sup> Since we do not consider Byzantine behaviors, issues related to information security and privacy are beyond the scope of this paper.

Processes exchange messages over a wireless radio network and all processes have the same radio transmission range. We assume the existence of a discrete global clock, i.e., the range  $T$  of the clock's ticks is the set of non-negative integers. We also assume the existence of a known bound on the relative speed of processes in the system. Finally, let  $p_i, p_j$  be two processes in  $P$ , we introduce the definitions given hereafter in order to capture proximity-based semantics.

- $loc_i(t)$  denotes the geographic location of mobile process  $p_i$  at time  $t \in T$ .
- $Z_i(\Delta_r, t)$  denotes all the points inside or on the circle centered at  $loc_i(t)$  with radius  $\Delta_r$ . We call  $Z_i(\Delta_r, t)$ , the *mobile circular zone* of radius  $\Delta_r$  around  $p_i$ . Thus, we use the term *mobile circular zone* to capture the notion of *neighborhood*.
- We say  $p_j$  is *in*  $Z_i(\Delta_r, t)$  if  $loc_j(t) \in Z_i(\Delta_r, t)$ .
- We say  $p_j$  *follows*  $p_i$  *within a radius*  $\Delta_r$  *during a time interval*  $[t_1, t_2]$ , with  $t_1, t_2 \in T$  and  $t_1 \leq t_2$ , if  $p_j$  remains permanently in the mobile circular zone of radius  $\Delta_r$  around  $p_i$  during the time interval. This can be expressed as follows:

$$\forall t \in [t_1, t_2] : loc_j(t) \in Z_i(\Delta_r, t) \quad (2.1)$$

If  $t_1 = t_2$ , stating that  $p_j$  *follows*  $p_i$  *within a radius*  $\Delta_r$  *during a time interval*  $[t_1, t_2]$  is equivalent to say that  $loc_j(t) \in Z_i(\Delta_r, t)$  at time  $t = t_1 = t_2$ .

- We say  $p_j$  *follows*  $p_i$  *within a radius*  $\Delta_r$  *after time*  $t$ , if  $p_j$  remains permanently in the mobile circular zone of radius  $\Delta_r$  around  $p_i$  after time  $t \in T$ . This can be expressed as follows:

$$\forall t' \in T : t' \geq t \Rightarrow loc_j(t') \in Z_i(\Delta_r, t') \quad (2.2)$$

- We say  $p_j$  *eventually follows*  $p_i$  *within a radius*  $\Delta_r$ , if there exists a time after which  $p_j$  remains permanently in the mobile circular zone of radius  $\Delta_r$  around  $p_i$ . This can be expressed as follows:

$$\exists t \in T : \forall t' \in T : t' \geq t \Rightarrow loc_j(t') \in Z_i(\Delta_r, t') \quad (2.3)$$

## 2.3 The Spotcast Abstraction

In this section, we introduce the *spotcast* communication abstraction in three variants. Intuitively, by using the spotcast service a mobile process can disseminate a message for a given time to all mobile processes located in its proximity. The name

“spotcast” is chosen by analogy with a spotlight following the source of the message (the spotcaster). Formally, the spotcast service supports the following functional interface:

- $\text{SPOTCAST}(m, \Delta_r, \Delta_t)$ : disseminates a message  $m$  in  $Z_i(\Delta_r, t)$  for all  $t \in E = [t_s, t_s + \Delta_t]$ , where  $p_i$  is the process that invokes the spotcast and  $t_s$  is the time when the spotcast is invoked.
- $\text{DELIVER}(m, p_i)$ : callback delivering a message  $m$  spotcast by process  $p_i$ .

The time interval  $E$  is called the *spotcast epoch*. All spotcast variants share a set of properties called the *core spotcast properties* listed hereafter.

### 2.3.1 Core Spotcast Properties

***Non-triviality.*** If some process  $p_j$  delivers a message  $m$  with sender  $p_i$ , then there exists a time  $t_E$  in  $E$  such that  $\text{loc}_j(t_E) \in Z_i(\Delta_r, t_E)$ .

***No Duplication.*** No message is delivered more than once.

***No Creation.*** If some process  $p_j$  delivers a message  $m$  with sender  $p_i$ , then  $m$  was previously spotcast by process  $p_i$ .

### 2.3.2 Spotcast Variants and their Validity Properties

In addition to the core properties, each spotcast variant satisfies a specific *validity* property. Hereafter, we present each variant and its *validity* property, together with an example of its usage (see Figure 2.1).

#### 2.3.2.1 Timely Spotcast

This variant provides the strictest *here-and-now* semantics among the three spotcast variants. Intuitively, it guarantees that there exists a time limit after which a message is delivered to all processes in the neighborhood of the process who spotcast that message. The corresponding formal validity property is given hereafter.



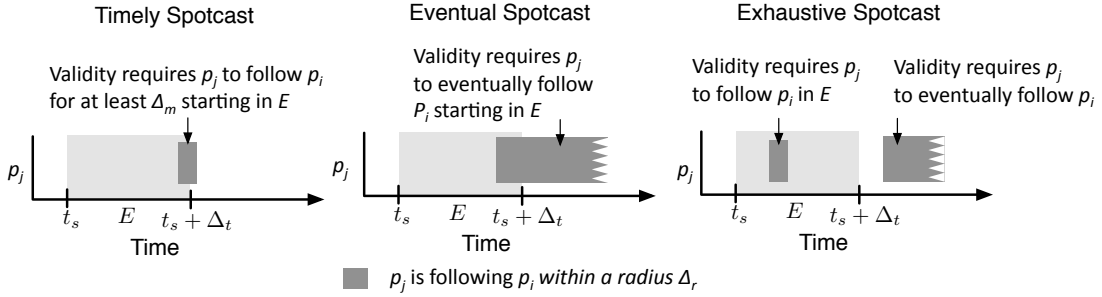


Fig. 2.1: Validity Properties of Spotcast Variants.

**Validity.** If a correct process  $p_i$  spotcasts a message  $m$ , there exists a bounded time duration  $\Delta_m$  such that every correct process  $p_j$  delivers  $m$  in  $E'=[t_s, t_s + \Delta_t + \Delta_m]$ , if  $p_j$  follows  $p_i$  within radius  $\Delta_r$  during  $[t_E, t_E + \Delta_m]$  with  $t_E$  in  $E$ .

The time interval  $E'=[t_s, t_s + \Delta_t + \Delta_m]$  is called *the delivery epoch*. The on-the-spot survey application discussed in Section 2.1 is an example of Timely Spotcast usage: the person launching such a survey is interested in disseminating questions to the audience in a timely manner.

### 2.3.2.2 Eventual Spotcast

This variant provides a somehow more elastic *here-and-now* semantics. Intuitively, it guarantees that any process staying long enough in the neighborhood of some process who is spotcasting a message, will eventually receive that message. The corresponding formal validity property is given hereafter.

**Validity.** If a correct process  $p_i$  spotcasts a message  $m$ , every correct process  $p_j$  eventually delivers  $m$ , if there exists a time  $t_E$  in  $E$  after which  $p_j$  follows  $p_i$  within the radius  $\Delta_r$ .

The social radar application discussed in Section 2.1 is an example of Eventual Spotcast usage: each user regularly spotcasts her location, so that users in her neighborhood will eventually learn about her nearby presence.

### 2.3.2.3 Exhaustive Spotcast

Finally, this variant provides the most elastic *here-and-now* semantics. Intuitively, it guarantees that any process who was in the neighborhood of some other process when the latter is spotcasting a message, will eventually receive that message, provided both processes eventually get the chance to stay close enough for long enough. The corresponding formal validity property is given hereafter.

**Validity.** If a correct process  $p_i$  spotcasts a message  $m$ , every correct process  $p_j$  eventually delivers  $m$ , if there exists a time  $t_E$  in  $E$  when  $loc_j(t_E) \in Z_i(\Delta_r, t_E)$  and also  $p_j$  eventually follows  $p_i$  within the radius  $\Delta_r$ .

The mobile photo sharing application discussed in Section 2.1 is an example of Exhaustive Spotcast usage: consider Alice, who is spotcasting a URL pointing to the photos she is taking while at a party. The exhaustive delivery property allows another participant in that party, say Bob, to receive the URL, even if Alice has to leave before Bob can actually receive the URL. The delivery will occur as soon as Alice and Bob get together again, possibly after the party, and stay in the neighborhood of each other long enough.

## 2.4 A Spotcast Algorithm

We provide an architecture overview of our algorithm for the spotcast communication abstraction in Figure 2.2. At the top, a mobile proximity-based application uses the spotcast service, which itself relies on two lower-level services for its implementation, namely a *global positioning service* and a *scoped broadcast service*.

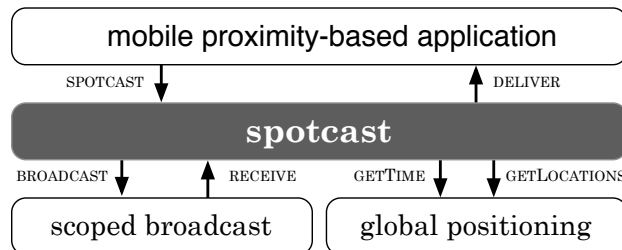


Fig. 2.2: Spotcast – Architecture Overview.

### 2.4.1 Global Positioning Service

This service allows each mobile process  $p_i$  to know its position in space and time, via the following functions:<sup>3</sup>

- **GETTIME**: returns the current global time. Formally, this implies that each process  $p_i$  has access to the global clock modeled in Section 2.2.
- **GETLOCATIONS**( $t_1, t_2$ ): returns the set of locations occupied by  $p_i$  during time interval  $[t_1, t_2]$ . If  $t_1 < \text{GETTIME}$  and  $t_2 > \text{GETTIME}$ , only locations occupied during time interval  $[t_1, \text{GETTIME}]$  are returned. If  $t_1 > \text{GETTIME}$  or if  $t_1 > t_2$ , no location is returned.

### 2.4.2 Scoped Broadcast Service

This communication service allows processes to send messages to all processes located within a given radius. Formally, the scoped broadcast service exposes the following primitives:

- **BROADCAST**( $m, \Delta_r$ ): broadcasts a message  $m$  in  $Z_i(\Delta_r, t_b)$ , where  $p_i$  is the sender and  $t_b$  is the time when the broadcast is invoked.
- **RECEIVE**( $m, p_i$ ): callback delivering a message  $m$  broadcast by process  $p_i$ .

In terms of safety, the scoped broadcast service satisfies the *no duplication* property and the *no creation* property given hereafter.

**No Duplication.** No message is delivered more than once.

**No Creation.** If some process  $p_j$  delivers a message  $m$  with sender  $p_i$ , then  $m$  was previously broadcast by process  $p_i$ .

When it comes to liveness, we consider two alternatives of the delivery property, namely *fair-loss delivery* and *timely delivery*; these two alternatives are presented below. As discussed in Section 2.4.4, the implementability of the various flavors of spotcast closely depends on which delivery property is being considered.

---

<sup>3</sup> In practice, such a service would typically be implemented using NASA's GPS or ESA's Galileo space-based satellite navigation technologies.

**Fair-Loss Delivery.** If a correct process  $p_i$  broadcasts a message  $m$  an infinite number of times, any correct process  $p_j$  eventually delivers  $m$ , if  $p_j$  eventually follows  $p_i$  within the radius  $\Delta_r$ .

**Timely Delivery.** If a correct process  $p_i$  broadcasts a message  $m$ , there exists a bounded time duration  $\Delta_{broadcast}$  such that every correct process  $p_j$  delivers  $m$  in interval  $[t_b, t_b + \Delta_{broadcast}]$ , if  $p_j$  follows  $p_i$  within the radius  $\Delta_r$  during  $[t_b, t_b + \Delta_{broadcast}]$ .

Note that it is beyond the scope of this paper to provide an implementation of scoped broadcast service. However, we discuss its implementability in Section 2.5.

### 2.4.3 Generic Spotcast Algorithm

Algorithm 2.1 presents our implementation of the spotcast communication abstraction. It is *generic* in the sense that it serves as basis for implementing any of the three variants introduced in Section 2.3. Its genericity is captured by function ISBOUNDEDBUFFER and by procedure BROADCAST. That is, depending on how they are defined, Algorithm 2.1 implements a specific variant of spotcast. Specifically, BROADCAST relies on an implementation of the Scoped Broadcast Service ensuring either *fair-loss delivery* or the *timely delivery*, while ISBOUNDEDBUFFER returns *true* if an implementation with bounded buffer is possible, *false* otherwise. Algorithm 2.1 can be divided in three parts, namely *initialization*, *spotcasting* and *delivering*, which are discussed hereafter.

1. *Initialization* (lines 3-5). The algorithm relies on two message sets: *msgset* for storing messages to be broadcast and *delivered* for storing the delivered spotcast messages.
2. *Spotcasting* (lines 6-17). Spotcasting entails a setup and a running phase. The setup phase (lines 6-13) starts when SPOTCAST primitive is called with following parameters: a message  $m$  to be spotcast in a radius  $\Delta_r$  during a time duration  $\Delta_t$ . This call creates a *msg* that encapsulates  $m$  and some other parameters. Among these parameters the hash map *locs* is used to store the location of the sender at each time  $t$  in the spotcast epoch. The *msg* is then added to the set *msgset*. The running phase (lines 14-17) periodically goes through each *msg* in *msgset*. Thus, all the locations of the process during the spotcast epoch are first assigned to the *msg*'s *locs*. Then, *msg* is broadcast within the *msg*'s radius using

---

**Algorithm 2.1** Generic Spotcast Algorithm at Process  $p_i$ 

---

```
1: implements: SPOTCAST
2: uses: BROADCAST, ISBOUNDEDBUFFER {generic procedure/function}

3: initialisation:
4:    $msgset \leftarrow \emptyset$ 
5:    $delivered \leftarrow \emptyset$ 

6: upon SPOTCAST( $m, \Delta_r, \Delta_t$ ) do
7:    $msg \leftarrow \perp$  {creates msg to encapsulate m plus some additional parameters}
8:    $msg.m \leftarrow m$  {assigns m}
9:    $msg.\Delta_r \leftarrow \Delta_r$  {assigns the radius}
10:   $msg.t_{start} \leftarrow \text{GETTIME}$  {assigns the start of the spotcast epoch}
11:   $msg.t_{end} \leftarrow msg.t_{start} + \Delta_t$  {assigns the end of the spotcast epoch}
12:   $msg.locs \leftarrow [ ]$  {locs is a hash map to store (t, loc_i(t)) for all t in the spotcast epoch}
13:   $msgset \leftarrow msgset \cup \{msg\}$  {adds msg to msgset}

14: do every  $\Delta_{period}$  for each  $msg$  in  $msgset$  {broadcasts periodically}
15:    $msg.locs \leftarrow \text{GETLOCATIONS}(msg.t_{start}, msg.t_{end})$ 
16:   trigger BROADCAST( $msg, msg.\Delta_r$ )
17:   if ISBOUNDEDBUFFER  $\wedge$  GETTIME  $>$   $msg.t_{end}$  then
18:      $msgset \leftarrow msgset \setminus \{msg\}$  {removes msg from the msgset}

19: upon RECEIVE( $msg, p_j$ ) do
20:   if LOCATIONMATCH( $msg$ )  $\wedge$   $msg.m \notin delivered$  then
21:     trigger DELIVER( $msg.m, p_j$ )
22:      $delivered \leftarrow delivered \cup \{msg.m\}$  {adds m to delivered messages}

23: function LOCATIONMATCH( $msg$ ) returns boolean is
24:   for all  $t \in [msg.t_{start}, msg.t_{end}]$  do
25:     if DISTANCE(GETLOCATIONS( $t, t$ ),  $msg.locs.get(t)$ )  $\leq msg.\Delta_r$  then
26:       return true {returns true if a location match occurred during the epoch}
27:   return false
```

---

the scoped broadcast service’s BROADCAST primitive. After the broadcast, if it is possible (line 17),  $msg$  is removed from  $msgset$ .

3. *Delivering* (lines 19-22). When a broadcast  $msg$  is received, RECEIVE callback of the underlying scoped broadcast is triggered. Then  $m$  of  $msg$  is delivered by the spotcast service via the DELIVER callback if it has not been already delivered and if the receiving process meets the requirement of a location match. This is done by calling the LOCATIONMATCH function. For a location match to occur (lines 23-27), the distance between the sender and the receiver must have been less than or equal to the  $msg.\Delta_r$  at least for some time during the spotcast epoch.

## 2.4.4 Correctness and Implementability

The *no creation* property of spotcast follows directly from the *no creation* property of the underlying scoped broadcast service. As for the *no duplication* property of spotcast, it follows from the *no duplication* property of scoped broadcast service and from the management of the *delivered* set (lines 19 to 22). The *non-triviality* property of spotcast is ensured by calling the function LOCATIONMATCH (lines 23 to 27) and delivering the message if and only if a location match occurs. Besides these three core properties, each spotcast variant comes with a distinct *validity* property. We separately discuss their respective correctness and implementability hereafter. Table 2.1 summarizes this discussion.

### 2.4.4.1 Validity of Timely Spotcast

Algorithm 2.1 implements Timely Spotcast if generic procedure BROADCAST guarantees *timely delivery*. Indeed, recall that a timely spotcast message  $m$  must be delivered in the *delivery epoch*  $E'=[t_s, t_s + \Delta_t + \Delta_m]$ , with  $\Delta_m$  bounded. Note also that we can express  $\Delta_m = \Delta_{period} + \Delta_{exec} + \Delta_{bcast}$ , where  $\Delta_{exec}$  is the execution time of (lines 14-17), and  $\Delta_{bcast}$  is the communication delay introduced by the underlying scoped broadcast.  $\Delta_{period}$  is a constant. Since there exists a known upper bound on the relative processing speed in the system, we already know that  $\Delta_{exec}$  is bounded. So for  $\Delta_m$  to be bounded, we only need to be sure that  $\Delta_{bcast}$  is also bounded. This is precisely what the timely delivery property guarantees. Now, since deliv-

ery is required only during bounded delivery epoch  $E'$ , a bounded message buffer is sufficient, i.e., function `ISBOUNDEDBUFFER` can return *true*, allowing *msgset* to be purged (line 18). On the other hand, Algorithm 2.1 can not implement Timely Spotcast if the procedure `BROADCAST` guarantees *fair-loss delivery*. The reason is that the fair-loss delivery property only guarantees an eventual message delivery.

#### 2.4.4.2 Validity of Eventual Spotcast

Algorithm 2.1 implements Eventual Spotcast if the generic procedure `BROADCAST` guarantees *fair-loss delivery* and if `ISBOUNDEDBUFFER` returns *false* (hence requires unbounded buffer). Indeed, Eventual Spotcast guarantees an eventual message delivery. This is offered by the fair-loss delivery property of the underlying scoped broadcast. Moreover, since fair-loss delivery requires the message to be broadcast infinitely often, an unbounded message buffer is necessary.

Algorithm 2.1 also implements Eventual Spotcast in the case that the generic procedure `BROADCAST` guarantees *timely delivery*. In this case however, a bounded message buffer is sufficient. Indeed, recall that the *validity* of Eventual Spotcast guarantees the message delivery to a correct  $p_j$  which follows the spotcaster  $p_i$  within the radius  $\Delta_r$  after  $t_E$ . This means that the latest possible time for  $p_j$  to start to follow  $p_i$  is  $t_E = t_s + \Delta_t$  (end of the spotcast epoch). According to the algorithm, if function `ISBOUNDEDBUFFER` returns *true*, the message *msg* containing the spotcast message *m* is broadcast for the last time after the termination of the *spotcast epoch* (lines 14 to 17). The timely delivery property guarantees the delivery of this last

Table 2.1: Spotcast – Correctness and Implementability.

<b>Spotcast</b>	<b>Scoped Broadcast</b>	
	<i>fair-loss delivery</i>	<i>timely delivery</i>
<i>timely</i>	not implementable	implementable with bounded buffer
<i>eventual</i>	implementable with unbounded buffer	implementable with bounded buffer
<i>exhaustive</i>	implementable with unbounded buffer	implementable with unbounded buffer

broadcast of  $msg$  to any  $p_j$  following  $p_i$  after  $t_E$  even for  $t_E = t_s + \Delta_t$ . This means that  $msg$  can be safely removed from  $msgset$  after its last broadcast which implies using a bounded buffer.

#### 2.4.4.3 Validity of Exhaustive Spotcast

Algorithm 2.1 implements Exhaustive Spotcast if the generic procedure BROADCAST guarantees *fair-loss delivery* or *timely delivery* and if ISBOUNDEDBUFFER returns *false* (hence requires unbounded buffer). Indeed, recall that the *validity* property of Exhaustive Spotcast guarantees an eventual message delivery to all correct processes which were within the radius  $\Delta_r$  for some time  $t_E$  in the *spotcast epoch*, and which eventually follow the spotcaster process  $p_i$  within the radius  $\Delta_r$ . Since the time when a process starts to eventually follow  $p_i$  is unknown,  $p_i$  should continue to broadcast the message infinitely often. This means that regardless of the underlying scoped broadcast service, Algorithm 2.1 can only implement Exhaustive Spotcast using an unbounded message buffer.

## 2.5 Implementability of Scoped Broadcast Service

The correctness of our spotcast algorithm relies on the existence of the scoped broadcast service, whose implementability in turn depends on the properties of the underlying MAC layer. So, in order to discuss the implementability of the scoped broadcast service over wireless ad hoc networks, we rely on the MAC layer models described in [8] and [18], which have been shown to realistically model the 802.11 MAC layer [1]. Since both these MAC layer models guarantee that no message is created or duplicated, the *no duplication* and the *no creation* properties of our scoped broadcast service are trivially ensured. So in the following, we focus on the implementability of the *delivery* property of each scoped broadcast variant. That is, we use the model described in [8] to show the implementability of the *fair-loss delivery* property, while the model described in [18] is used to show the implementability of the *timely delivery* property. For each delivery property, the discussion distinguishes two cases: (1) the *single-hop* case, where the radio transmission range is greater than or equal to the  $\Delta_r$  radius of the scoped broadcast, and (2) the *multi-hop* case, where the radio transmission range is smaller than  $\Delta_r$ .



## 2.5.1 Implementability of the Fair-Loss Delivery Property

In [8], the MAC Layer model assumes that the communication medium is prone to collisions but guarantees *an eventual collision freedom* property. In addition, node clock skews and inter-node communication delays are assumed to be bounded by known constants. Processing is then conceptually divided into synchronous rounds and at each round each node broadcasts at most one message. Nodes can fail by crashing but cannot crash while broadcasting. A node that does not crash throughout an entire execution is said to be *correct*.

### 2.5.1.1 Single-hop Case

The *fair-loss delivery* property guarantees an *eventual* delivery of a message if it is broadcast infinitely often. In the single-hop case, one way to achieve the *fair-loss delivery* property is to assume that eventually the communication medium becomes collision-free.<sup>4</sup> This is precisely the property ensured by the *eventual collision freedom* property of the considered MAC layer. According to this property, there exists a positive integer  $b$ , such that in each execution, there exists a round  $r_{ecf}$ , so that for every round  $r \geq r_{ecf}$ , if at most  $b$  nodes broadcast in  $r$ , then every message broadcast in  $r$  is reliably delivered by all correct nodes. To eventually reach a round in which at most  $b$  nodes broadcast, a *wake-up service* is used. This service reduces contention by determining which nodes should broadcast in a given round. The wake-up service eventually stabilizes, i.e., it eventually recommends that at least one, and no more than  $b$ , correct nodes can broadcast in a round. Thus, in executions which satisfy the *eventual collision freedom* property, this allows the messages to be delivered without collision. The wake-up service can be implemented by using a simple approximation of a well-known back-off strategy [12; 32; 31; 35].

### 2.5.1.2 Multi-hop Case

When the radio transmission range is smaller than the radius  $\Delta_r$  of the scoped broadcast, a scoped broadcast protocol can guarantee the *fair-loss delivery* property if the two following conditions are satisfied: a) the underlying MAC layer satisfies the *fair-loss delivery* property and b) a route is established between each correct

---

<sup>4</sup> We assume that collisions are the major source of message losses.

node  $p_i$  and any other correct node  $p_j$  which is in radius  $\Delta_r$  around  $p_i$ . As just discussed, the MAC layer modeled in [8] satisfies the *fair-loss delivery* property, so Condition a) can be fulfilled by using such a MAC layer. For Condition b) to hold, the scoped broadcast can use for instance a *proactive* routing protocol (e.g., [33]) as sub-protocol, provided of course that there exists a path between  $p_i$  and  $p_j$  where the distance between any two direct neighbors along the path does not exceed the transmission range.

## 2.5.2 Implementability of the Timely Delivery Property

To discuss the implementability of the *timely delivery* property, we rely on the MAC layer modeled in [18]. Intuitively, this MAC layer provides the ability to reliably broadcast messages and to receive acknowledgments when those messages have been successfully delivered to all nearby nodes, *with timing guarantees*. In practice, such a MAC layer is implementable with very high probability<sup>5</sup>, by using contention-management mechanisms such as carrier sensing and back-off [1], receiver-side collision detection with negative acknowledgment [9], or network coding methods [13].

### 2.5.2.1 Single-hop Case

An implementation of the *timely delivery* property in the single-hop case must guarantee a finite upper bound  $\Delta_{bcast}$  on the duration between the broadcast and the delivery of a message, provided the receiver remains in the transmission range of the sender during  $\Delta_{bcast}$ . This can be ensured thanks to the *guaranteed communication* property of the discussed MAC layer. This property can be stated as follows: let process  $p_i$  be the sender of a message  $m$ , then any process  $p_j$  which remains in the communication range of  $p_i$  for all time between the broadcast and the acknowledgment of  $m$ , receives  $m$  before the acknowledgment of  $m$  at  $p_i$ . In addition, the *guaranteed communication* property comes with a timing guarantee on  $m$ 's reception, defined as function  $F_{ack}^+$ . This function expresses an upper bound on the elapsed time between the broadcast of  $m$  and the reception of its acknowledgment at sender  $p_i$  in the worst case, which corresponds to the maximum contention level. The contention level is defined as the number of distinct senders in the neighborhood

---

<sup>5</sup> In practice, synchronous assumptions are always probabilistic.

of  $m$ 's receivers and  $m$ 's sender, and whose broadcast overlaps the broadcast and acknowledgment of  $m$ . Thus, assuming  $\Delta_{broadcast} = F_{ack}^+$ , the *timely delivery* property is trivially ensured by the considered MAC layer for the single-hop case.

### 2.5.2.2 Multi-hop Case

When the radio transmission range is smaller than the radius  $\Delta_r$  of the scoped broadcast, a scoped broadcast protocol can guarantee the *timely delivery* property if the two following conditions are satisfied: a) the underlying MAC layer satisfies the *timely delivery* property and b) a route is established between each correct node  $p_i$  and any other correct node  $p_j$  which is in radius  $\Delta_r$  around  $p_i$ . As just discussed, the MAC layer modeled in [18] satisfies the *timely delivery* property for the single-hop case with  $\Delta_{broadcast} = F_{ack}^+$ , so Condition a) is fulfilled. For Condition b), we can follow the same reasoning we did in Section 2.5.1.2 for the *fair-loss delivery* property: to find a route between any pair of correct nodes  $p_i$  and  $p_j$ , provided there exists a path between  $p_i$  and  $p_j$ . In addition, we must show that the routing of any message is time bounded. For this, let  $d(i, j)$  be the length of the shortest path between any two pairs of correct nodes  $p_i$  and  $p_j$ , and let  $k$  be the maximum of such  $d(i, j)$  lengths. By observing that  $\Delta_{broadcast} = O(k \times F_{ack}^+)$  for the considered MAC layer, we can say that the *timely delivery* property can be ensured for the multi-hop case.

## 2.6 Related Work

Among location-aware or time-aware communication services proposed for mobile ad hoc networks, *geocast* and *mobicast* are the closest to our work. In this section, we discuss these services and compare them to spotcast. We then compare spotcast to higher level services such as *location-based publish/subscribe*.

### 2.6.1 Geocast

In a geocast routing protocol, a message is disseminated to all nodes which are within a given geographic area called the *geocast region*. Thus, geocast is a type of multicast in which group membership is defined with respect to a geographic area. Geocast

was initially proposed for the Internet [19]. Then, various geocast routing protocols were proposed for ad hoc networks [21; 22; 4; 25; 24; 29; 28; 3] and in particular, for vehicular ad hoc networks (VANETs) [2; 20; 34; 26; 27]. The classical geocast routing is semantically time-oblivious i.e., the geocast message is assumed to be delivered as soon as possible. In *abiding geocast* [29], the geocast message is disseminated in the geocast region for a time duration. Some papers use abiding geocast to disseminate traffic warning messages or commercial advertisement to a group of vehicles in a given zone for a given time [34; 26]. There exist several differences between geocast and spotcast, depending on the geocast variant. With spotcast, the zone in which the message is disseminated is a disk centered at the source of the message and that zone moves with the source of the message. With geocast on the contrary, the dissemination region is generally not associated with the source of the message and remains stationary. Furthermore, the delivery guarantees of our different spotcast variants require the receiver to follow the source within some range for some amount of time, whereas such a requirement is absent from the geocast specification.

## 2.6.2 Mobicast

Mobicast is a class of multicast which is both location-aware and time-aware. In mobicast, a message is disseminated in an area called the *delivery zone* for a time duration  $T$ . Delivery zone can move during  $T$  and is denoted as  $Z[t]$ , where  $t$  is in  $T$ . As the delivery zone moves, some nodes enter the zone and some nodes leave the zone. The ultimate goal of mobicast is to achieve *just-in-time* message delivery, i.e.,  $Z[t]$  represents the area where the mobicast message should be delivered at time  $t$  [16]. Some mobicast protocols were proposed for wireless sensor networks [16; 6; 17] and VANETs [7]. In some of these protocols, the delivery zone is not associated to the source of the message e.g., in [16]. In [7], the delivery zone is an elliptic area around the source of the message (a moving car). Contrary to spotcast, mobicast does not require the nodes to follow the message source in order to guarantee the message delivery. Instead, mobicast protocols usually assume the existence of a *forwarding zone* to ensure the implementation of the strong *just-in-time* message delivery property.

### 2.6.3 Location-Based Publish/Subscribe

Some authors proposed high level communication abstractions derived from the publish/subscribe paradigm, which include some type of constraint on messages in time and/or space [15; 30; 11]. In STEAM [30], messages are constrained to a location around the sender. However, there is no support for persistence and the specifications and underlying communication abstractions are not detailed. In [11], messages are persisted and can be localized. However, they are assigned to a fixed geographical zone and do not move around with the sender. In [15], messages are persisted in a geographical zone around the publisher for a certain duration. The communication abstraction used in [15] to propagate messages is the closest to spotcast, however it does not provide any guarantees. This lack of guarantees is one of the main motivations for the present work.

## 2.7 Conclusion

In this paper, we presented spotcast, a communication abstraction specifically devised for proximity-based mobile applications. Spotcast allows processes to send messages to peers located within a defined range in a defined time frame. We presented three variants of spotcast offering different levels of delivery guarantees. By proposing an interface and formal properties for a novel communication abstraction in MANETs, this paper aims at filling a current gap in communication support for emerging proximity-based mobile applications. However, several issues remain open, which we intend to address in future work. For instance, we will consider Byzantine processes in order to model security and privacy issues, which are inherent to proximity-based mobile applications.

Moreover, the partial requirement for unbounded buffers needs to be further addressed from a realistic, application-driven perspective. For instance, a discussion on what might be the finite size of a buffer to still be considered as unbounded depending on application might be useful. Another issue which could be investigated is the feasibility of spotcast variants, particularly exhaustive spotcast, based on real applications, the network size and the number of broadcasting nodes.

## References

- [1] IEEE 802.11: Wireless LAN MAC and Physical Layer Specifications. June (1999).
- [2] Bachir, A., Benslimane, A.: A Multicast Protocol in Ad hoc Networks Inter-vehicle Geocast. In: Proceedings of IEEE Semiannual Vehicular Technology Conference, VTC'03, 4:2456–2460, IEEE (2003).
- [3] Baldoni, R., Ioannidou, K., Milani, A.: Mobility Versus the Cost of Geocasting in Mobile Ad-hoc Networks. DISC, Lecture Notes in Computer Science, 4731:48-62 (2007).
- [4] Boleng, J., Camp, T., Tolety, V.: Mesh-Based Geocast Routing Protocols in an Ad hoc Network. In: Proceedings of 15th International Parallel and Distributed Processing Symposium, IPDPS '01, pp. 1924–1933, IEEE Comput. Soc. (2001).
- [5] Chandra, T. D., Toueg, S.: Unreliable Failure Detectors for Reliable Distributed Systems. JACM., 43:225–267 (1996).
- [6] Chen, Y.S., Ann, S.Y., Lin, Y.W.: VE-Mobicast: A Variant-Egg-Based Mobicast Routing Protocol for SensorNet. ACM Wireless Networks., 14:199–218 (2008).
- [7] Chen, Y.S., Lin, Y.W., Lee, S.L.: A Mobicast Routing Protocol in Vehicular Ad-hoc Networks. Mob. Netw. Appl., 15:20–35 (2010).
- [8] Chockler, G., Demirbas, M., Gilbert, S., Newport, C., Nolte, T.: Consensus and Collision Detectors in Wireless Ad hoc Networks. In: Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, PODC'05, pp. 197–206, ACM (2005).
- [9] Chockler, G., Demirbas, M., Gilbert, S., Lynch, N., Newport, C., Nolte, T.: Consensus and Collision Detectors in Radio Networks. Distrib. Comput., 21, 55–84 (2008).
- [10] Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the Presence of Partial Synchrony. JACM., 35:288–323 (1988).
- [11] Frey, D., Roman, G.: Context-Aware Publish Subscribe in Mobile Ad hoc Networks. In: Proceedings of the 9th International Conference on Coordination Models and Languages, Coordination'07, pp. 37–55, Springer (2007).
- [12] Gallager, R.: A perspective on Multiaccess Channels. IEEE Transactions on Information Theory, 31(2):124–142 (1985).
- [13] Gollakota, S., Katabi, D.: ZigZag Decoding: Combating Hidden Terminals in Wireless Networks. In: Proceedings of the ACM SIGCOMM Conference (2008).

- [14] Hadzilacos, V., Toueg, S.: Fault-Tolerant Broadcasts and Related Problems. pp. 97–145, ACM Press/Addison-Wesley Publishing Co., New York (1993).
- [15] Holzer, A., Eugster, P., Garbinato, B.: Evaluating Implementation Strategies for Location-based Multicast Addressing. *IEEE TMC* (to appear) (2012).
- [16] Huang, Q., Lu, C., Roman, G.: Mobicast : Just-in-Time Multicast for Sensor Networks under Spatiotemporal Constraints. In: Proceedings of the International Conference on Information Processing in Sensor Networks, IPSN’03, pp. 442–457 (2003).
- [17] Huang, Q., Lu, C., Roman, G.: Reliable Mobicast via Face-Aware Routing. In: Proceedings of IEEE International Conference on Computer Communications, INFOCOM’04, pp. 205–217 (2004).
- [18] Kuhn, F., Lynch, N., Newport, C.: The Abstract Mac Layer. *Distributed Computing*, 24(3-4):187–206 (2011).
- [19] Imielinski, T., Navas, J.C.: Gps-Based Geographic Addressing, Routing, and Resource Discovery. *Commun. ACM.*, 42:86–92 (1999).
- [20] Joshi, H.P., Sichitiu, M., Kihl, M.: Distributed Robust Geocast Multicast Routing for Inter-vehicle Communication. In: Proceedings of Weird Workshop on WiMAX, Wireless and Mobility, Weird’07, pp. 9–21 (2007).
- [21] Ko, Y.B., Vaidya, N.H.: Geocasting in Mobile Ad hoc Networks: Location-Based Multicast Algorithms. In: Proceedings of Second IEEE Workshop on Mobile Computing Systems and Applications, WMCSA’99, pp. 101–110, IEEE (1999).
- [22] Ko, Y.B., Vaidya, N.H.: Geotora: a Protocol for Geocasting in Mobile Ad hoc Networks. In: Proceedings of International Conference on Network Protocols, ICNP’00, (00-010):240–250, IEEE Comput. Soc. (2000).
- [23] Lamport, L., Shostak, R., Pease, M.: The Byzantine Generals Problem. *ACM. TOPLAS.*, 4:382–401 (1982).
- [24] Lee, S.H., Ko, Y.B.: Geometry-Driven Scheme for Geocast Routing in Mobile Ad Hoc Networks. In: Proceedings of the IEEE Vehicular Technology Conference, VTC’06, pp. 638–642, IEEE (2006).
- [25] Liao, W. H., Tseng, Y.C., Lo, K.L. , Sheu, J. P.: GeoGRID: A Geocasting Protocol for Mobile Ad Hoc Networks Based on GRID. *JIT.*, 1–2:23–32 (2000).
- [26] Lim, Y., Ahn, S., Cho, K.H.: Abiding Geocast for Commercial Ad Dissemination in the Vehicular Ad hoc Network. In: Proceedings of IEEE International Conference on Consumer Electronics, ICCE’11, pp. 115–116 (2011).

- [27] Lin, Y., Chen, Y., Lee, S.: Routing Protocols in Vehicular Ad hoc Networks: A Survey and Future Perspectives. *J. Inf. Sci. Eng.*, 26(3):913–932 (2010).
- [28] Maihöfer, C.: A Survey of Geocast Routing Protocols. *IEEE Communications Surveys and Tutorials*, 6(1-4):32–42 (2004).
- [29] Maihöfer, C., Leinmüller, T., Schoch, E.: Abiding Geocast: Time–Stable Geocast for Ad hoc Networks. In: *Proceedings of the 2nd ACM International Workshop on Vehicular Ad hoc Networks, VANET '05*, pp. 20–29, ACM (2005).
- [30] Meier, R., Cahill, V.: On Event-Based Middleware for Location-Aware Mobile Applications. *IEEE TSE*, 36(3):409–430 (2010).
- [31] Nakano, K., Olariu, S.: Uniform Leader Election Protocols in Radio Networks. In: *Proceedings of the International Conference on Parallel Processing, ICPP'01*, pp. 240–250. IEEE Computer Soc. (2001).
- [32] Olariu, S., Nakano, K.: A survey on Leader Election Protocols for Radio Networks. In: *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks, ISPAN'02*, pp. 71–79, IEEE Computer Soc. (2002).
- [33] Perkins, C.E., Bhagwat, P.: Highly dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In: *Proceedings of ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'94)*, vol. 24, no.4, pp.234-244, (1994).
- [34] Yu, Q., Heijenk, G.: Abiding Geocast for Warning Message Dissemination in Vehicular Ad hoc Networks. In: *Proceedings of International Conference On Communications, ICC'08*, pp. 400–404 (2008).
- [35] Willard, D. E.: Log-logarithmic Selection Resolution Protocols in a Multiple Access Channel. *SIAM Journal of Computing*, 15(2):468–477 (1986).



**Part II**

**Abstractions and Algorithms for  
Proximity-Based Neighbor Detection**



## Chapter 3

# Effective and Efficient Neighbor Detection for Proximity-Based Mobile Applications

**Abstract** We consider the problem of maximizing both *effectiveness* and *efficiency* of the detection of a device by another device in a mobile ad hoc network, given a maximum amount of time that they remain in the proximity of each other. Effectiveness refers to the degree to which the detection is successful, while efficiency refers to the degree to which the detection is energy saving. Our motivation lies in the emergence of a new trend of mobile applications known as proximity-based mobile applications which enable a user to communicate with other users in some defined range and for a certain amount of time. The highly dynamic nature of these applications makes neighbor detection time-constrained, i.e., even if a device remains in proximity for a limited amount of time, it should be detected with a high probability as a neighbor. In addition, the limited battery life of mobile devices requires the neighbor-detection to be performed by consuming as little energy as possible. To address this problem, we perform a realistic simulation-based study in mobile ad hoc networks and we consider three typical urban environments where proximity-based mobile applications are used, namely *indoor with hard partitions*, *indoor with soft partitions* and *outdoor urban areas*. In our study, a node periodically broadcasts a message in order to be detected as a neighbor. Thus, we study the effect of parameters that we believe could influence effectiveness and efficiency, i.e., *the transmission power* and *the time interval between two consecutive broadcasts*. Our results show that regardless of the environment, effectiveness and efficiency are in conflict with each other. Thus, we propose a metric that can be used to make good tradeoffs between effectiveness and efficiency.

## Publication:

B. Bostanipour and B. Garbinato, Effective and efficient neighbor detection for proximity-based mobile applications, In *Elsevier Computer Networks Journal*, vol. 79, pp. 216–235, 2015.

The primary version of this work is published as a conference paper:

B. Bostanipour and B. Garbinato, Improving neighbor detection for proximity-based mobile applications, In *Proceedings of The 12th IEEE International Symposium on Network Computing and Applications (IEEE NCA'13)*, pp. 177–182, Cambridge, Massachusetts, USA, IEEE, 2013.

## 3.1 Introduction

With the increasing use of mobile devices and particularly smartphones, we face the emergence of a new blend of distributed applications known as *Proximity-Based Mobile (PBM)* applications [10; 11; 12]. These applications enable a user to interact with others in a defined range and for a given time duration e.g., for social networking (WhosHere [53], LoKast [31]), gaming (Bluetooth gaming apps [8]) and driving (Waze [52]).

Discovering who is nearby is a basic requirement of various PBM applications. In a simple usage scenario of social networking applications such as WhosHere [53] or LoKast [31], a user can discover other users in a defined range, view their profiles and chat with a user or a group of users with her phone. Usually, the highly dynamic nature of these applications (which is basically due to the mobility of devices) makes neighbor detection time-constrained, i.e., even if a device remains in proximity for a limited amount of time, it should be detected with a high probability as a neighbor. In addition, the limited battery life of mobile devices requires the neighbor-detection to be performed by consuming as little energy as possible.

In this paper, we consider the following problem: *how can a device be detected by another device with both maximum effectiveness and maximum efficiency, given a maximum amount of time that they remain in proximity of each other? If not, how can an effectiveness-efficiency tradeoff be made?* Effectiveness refers to the degree to which the detection is successful and is measured by the detection probability, while efficiency refers to the degree to which the detection is energy saving and is

measured by the inverse of energy consumption per device. To address this problem, we evaluate effectiveness and efficiency in a single-hop mobile ad hoc network (MANET). The evaluations are performed under realistic assumptions and based partly on simulations using the ns-2 [37] network simulator.<sup>1</sup>

There are two main reasons behind our choice of a MANET as the underlying network architecture. Firstly, MANETs seem to be the most natural existing technology to enable PBM applications. In fact, similarly to PBM applications, in a MANET two nodes can communicate if they are within a certain distance of each other (to have radio connectivity) for a certain amount of time. Secondly, mobile devices are increasingly equipped with ad hoc communications capabilities (e.g., WiFi in ad hoc mode or Bluetooth) which increases the chance of MANETs to be one of the future mainstream technologies for PBM applications.

Since the quality of radio signals (and consequently the detection probability) is affected by the environment attenuation, for our study we consider three typical urban environments where PBM applications are used, i.e., *indoor with hard partitions* (corresponding to offices with thick walls), *indoor with soft partitions* (corresponding to exhibitions with temporary partitions) and *outdoor urban areas* (corresponding to a music festival in downtown). To simulate these environments, we use a radio propagation model known for modeling the obstructed urban environments called *Log-Normal Shadowing* (LNS).

In our study, a node periodically broadcasts a *hello* message during a fixed time interval in order to be detected as a neighbor. We assume that the nodes use the *IEEE 802.11a* standard for the physical and mac layer. Thus, we study the impact of two key parameters that influence effectiveness and efficiency, i.e., *the transmission power* and *the time interval between two consecutive broadcasts*. In performing the evaluations, we are particularly interested to answer the following questions:

- *In each environment, when does a change in the value of any of the above mentioned parameters increase effectiveness and efficiency, or on the contrary, when does it deteriorate them?*
- *In each environment, is there a unique combination of these parameters that could maximize both effectiveness and efficiency? If not, how could a tradeoff between effectiveness and efficiency be made?*

---

<sup>1</sup> The version 2.35 (the latest version), released on November 4, 2011.

### 3.1.1 Contributions and Roadmap

This paper is, to the best of our knowledge, the first study on the impact of transmission power and broadcast interval on effectiveness and efficiency of neighbor detection for MANETs in urban environments. It provides a detailed simulation study and defines the metrics that can be used to interpret the results. In order for our results to be close to reality, the study is performed under realistic assumptions. For one thing, we use 802.11a technology for communication between nodes and we assume a probabilistic radio propagation model for urban environments. Furthermore, we calculate the energy consumption using the specification of typical smartphones.

The remainder of the paper is as follows. In Section 3.2, we describe our system model. In particular, we define the *neighbor detection algorithm*, which takes transmission power and broadcast interval (this pair constitutes a *strategy*) as input. In Section 3.3, we formulate the problem studied in this paper. It basically consists of finding the most effective and the most efficient strategy in each environment. If these strategies are not equal in an environment, we intend to find a strategy that makes a reasonable tradeoff between effectiveness and efficiency. We also define the set of strategies for which the effectiveness and efficiency are evaluated. In Section 3.4, we evaluate the effectiveness for the set of predefined strategies. We also discuss the impact of changing transmission power and broadcast interval on effectiveness. Finally, we identify the most effective strategy in each environment. In Section 3.5, we evaluate the efficiency for the set of predefined strategies. We show that efficiency is independent of the environment and we discuss the impact of changing transmission power and broadcast interval on efficiency. Finally, we identify the most efficient strategy. In Section 3.6, we compare the results of Sections 3.4 and 3.5. We observe that we cannot find a strategy that maximizes both effectiveness and efficiency in any environment. The reason is that, regardless of environment, effectiveness and efficiency are in conflict with each other. We then propose an approach to make a tradeoff between effectiveness and efficiency. Using this approach, we find the tradeoff strategy in each environment and we show that it has a relatively good effectiveness and efficiency compared to other strategies. Finally, we discuss related work in Section 3.7 before concluding in Section 3.8 with a perspective on future work.

## 3.2 System Model

In this section, we present the system model, and whenever necessary, we describe the reasons behind our modeling choices.

### 3.2.1 Processes

We consider a mobile ad-hoc network (MANET) consisting of a finite set of  $n$  processes  $P = \{p_1, \dots, p_n\}$ . We use the terms *process* and *node* interchangeably. Processes are in a two-dimensional plane. Each process has a unique identifier and is aware of its own geographic location at any time. Processes can experience *crash* failures. A crash faulty process stops prematurely. Prior to stopping, it behaves correctly. Since we do not consider *Byzantine* behaviors, information security and privacy issues are beyond the scope of this paper.

### 3.2.2 Time

We assume the existence of a discrete global clock, i.e., the clock's tick range is the set of non-negative integers. Every process has a local clock which has the same clock's tick range as the global clock and runs at the same rate as the global clock, but its time value has some offset from the global time.

### 3.2.3 Communication

We consider a single-hop network i.e., without any message routing mechanism. Processes communicate by broadcasting messages using the *IEEE 802.11a* MAC and physical layers [2; 28]. The current WiFi technology used in mobile devices is based on three IEEE standards, i.e., *802.11a*, *802.11b*, *802.11g*. There are two main reasons for our choice of 802.11a over the other standards: (1) 802.11b/g operate in the 2.4 GHz frequency band which is heavily used not only by WiFi devices but also by other devices such as microwave ovens and DECT phones whereas, 802.11a operates in the relatively unused 5 GHz frequency band. Thus, using 802.11a results in less

interference and better throughput. This makes 802.11a an appealing technology for ad hoc communication in urban areas where PBM applications are mostly used; (2) the most recent IEEE 802.11 standard, i.e., *802.11ac* also operates in 5 GHz frequency band [3] and uses some similar modulation schemes and coding rates for broadcast as 802.11a. Thus, using 802.11a allows us to have the results which are close to those that could be obtained with the new standard.

Finally, we assume that each process has a buffer (a queue) that stores messages after their generation and before their broadcast. The size of the queue and the messages are such that the queue never remains full long enough to cause it to drop a message.

### 3.2.4 Environment

We consider three typical urban environments where PBM applications are used, namely *indoor with hard partitions* (corresponding to offices with thick walls), *indoor with soft partitions* (corresponding to exhibitions with temporary partitions) and *outdoor urban areas* (corresponding to a music festival in downtown). In our study, we use a probabilistic model called the *Log-Normal Shadowing (LNS)* for the radio propagation in an urban environment [40]. LNS uses a log-normal random variable to describe the variations of the received power and has two parameters, i.e., *the path loss exponent* ( $\beta$ ) and *the shadowing deviation* ( $\sigma$ ) to characterize each environment. The path loss exponent ( $\beta$ ) captures the average signal attenuation due to effects such as absorption, refraction, diffraction, reflection, etc. The shadowing deviation ( $\sigma$ ) captures the radio irregularity. If ( $\sigma = 0$ ), the radio propagation range is a perfect circle, but as  $\sigma$  grows, its shape changes from a circle to a more random and irregular shape which reflects what happens in reality, i.e., in the presence of not perfectly isotropic antennas and the obstacles that cut the transmission range [36]. For our

Table 3.1: Values of LNS Parameters for each Environment.

Environment	$\beta$	$\sigma$ (dB)
Indoor-hard partitions	5.5	7
Indoor-soft partitions	5	9.6
Outdoor-urban	4	5.5



evaluations, we consider a distinct pair of  $(\beta, \sigma)$  values for each environment (see Table 3.1). These pairs are chosen based on the measurements in the literature [40].

### 3.2.5 Neighbor Detection Algorithm

Each process  $p_i$  executes the *neighbor detection algorithm*. The algorithm has two input parameters: the time duration  $\Delta_{period}$  and the transmission power  $pow_{tx}$ . There is also a constant  $R$  which defines the detection range. The algorithm divides time into rounds of  $\Delta_{period}$ . At the beginning of each round,  $p_i$  broadcasts a *hello* message containing the tuple  $(i, roundNo, loc)$  where *roundNo* is the number of the current round and *loc* is the location of  $p_i$  at time when *hello* is sent.

When a process  $p_j$  receives a *hello* message sent by  $p_i$ , it verifies if its distance to  $p_i$  is less than or equal to  $R$ . If it is the case,  $p_i$  is detected as a neighbor at its round *roundNo* by  $p_j$ . This means that if  $p_i$  is in the neighborhood of  $p_j$  since its first round of broadcasting the *hello* message, we can say that  $p_i$  is detected after being in the neighborhood of  $p_j$  for time duration of  $roundNo \times \Delta_{period}$ . Note that here we ignore the elapsed time between the sending and the reception of the *hello* message, which obviates the use of a time synchronization algorithm. Also, since we consider a single-hop network,  $p_j$  can only detect  $p_i$  as a neighbor if  $R$  is smaller than or equal to the actual transmission range of  $p_i$ .

In what follows, for simplicity's sake, we designate by the term *strategy* an ordered pair  $(pow_{tx}, \Delta_{period})$  which can be considered as a possible input of the neighbor detection algorithm.

## 3.3 Problem Statement

We characterize neighbor detection by two main aspects:

- *Effectiveness* is defined as the degree to which the neighbor detection is successful and is measured by the detection probability. Thereby, maximizing effectiveness boils down to maximizing the detection probability.
- *Efficiency* is defined as the degree to which the detection is energy saving and is measured by the inverse of energy consumption per process. Thereby, maximizing efficiency boils down to minimizing the energy consumption per process.

Thus, the problem we consider in this paper can be specified as follows. Let  $\Delta_{neighborhood}$  be the maximum amount of time that a node  $p_i$  remains continuously within the detection range  $R$  of a node  $p_j$ , then, for each environment, our goal is to find:

- *the most effective strategy*, i.e., the strategy that maximizes the effectiveness of detection of  $p_i$  by  $p_j$ ;
- *the most efficient strategy*, i.e., the strategy that maximizes the efficiency of detection during  $\Delta_{neighborhood}$ ;
- *the tradeoff strategy*, i.e., the strategy that makes a tradeoff between effectiveness and efficiency in the case that the most effective strategy is not the same as the most efficient strategy.

We address the problem by evaluating the effectiveness and the efficiency for a set  $S$  of predefined strategies and  $\Delta_{neighborhood} = 4$  seconds.

Since the majority of current PBM applications run on smartphones, for defining the strategies in  $S$  and later in our evaluations, we use the characteristics of current smartphones. For instance, regarding power consumption, we use the specifications of Broadcom’s wireless network interface cards [13]. In fact, in a mobile device, a *wireless network interface card* (denoted by *WNIC*) is the component that implements the MAC and the physical layers of the OSI model. Broadcom’s WNICs are one of the most used WNICs in the current mobile devices and specially smartphones, e.g., Broadcom’s BCM 4330 WNIC is used in both Samsung Galaxy S II and iPhone 4S [4].

Thus, let strategy  $s = (pow_{tx}, \Delta_{period})$  be an element of  $S$ . Then,  $pow_{tx}$  can take a value of 15 dBm, 19 dBm or 25 dBm. The first two values are based on specifications of Broadcom’s BCM 4329 and BCM 4330 WNICs, whereas the last value presents the possible performance gain of more powerful radio transmitters [49]. Also,  $\Delta_{period}$  can take a value of 1 second, 1/2 second, 1/4 second or 1/12 second. These values have been selected after our preliminary tests which show that they can present our results in a useful manner. Considering all the combinations of the above mentioned values of  $pow_{tx}$  and  $\Delta_{period}$ , the set  $S$  contains 12 strategies.

The reason behind the choice of 4 seconds for  $\Delta_{neighborhood}$  is that current PBM applications usually guarantee the detection of a person even if she remains in neighborhood for a very limited amount of time. Therefore, we choose  $\Delta_{neighborhood}$  to be reasonably short.

We also assume that each *hello* message has a size of 500 bytes. This value is chosen by considering the possibility that each message can be digitally signed and accompanied by a certificate to authenticate the sender (i.e., using a similar mechanism for message authentication as the one used for safety messages in vehicular ad hoc networks [41]). As stated earlier, in this paper we do not study the security and privacy issues. However, we choose the messages to be large enough so that our results remain valid for more general cases.

### 3.4 Evaluation of Effectiveness

Effectiveness is measured by the neighbor detection probability. Thus, in this section we first describe our approach to calculate the detection probability of each strategy, which is based on simulations. Then, we present our simulation setup and the results. In particular, while presenting the results, we discuss how a change in  $pow_{tx}$  or  $\Delta_{period}$  can affect the detection probability in each environment. We also define two packet dropping metrics that we use to interpret the results. Finally, we compare the effectiveness of the strategies and present the most effective strategy in each environment.

#### 3.4.1 Approach to Calculate the Neighbor Detection Probability

We calculate the resulting detection probability of each strategy by performing simulations with ns-2 simulator. Each simulation takes a strategy  $s$  and an environment  $e$  as the input and produces as the output the detection probability at time  $t$  for all  $t \in [0, \Delta_{neighborhood}]$ .

More precisely, let  $s = (pow_{tx}, \Delta_{period})$  and  $e = (\beta, \sigma)$ , at the beginning of a simulation we initialize the *neighbor detection algorithm* at all nodes with  $pow_{tx}$  and  $\Delta_{period}$ . We also initialize the radio propagation model with  $\beta$  and  $\sigma$ . Since the value of  $\Delta_{neighborhood}$  is the same while testing different strategies, instead of using nodes with movement, we consider static nodes which broadcast the *hello* message only during  $\Delta_{neighborhood}$ . However, since each node's local clock has some offset with respect to the global clock, nodes do not start and finish broadcasting

at the same time. We only verify the detection probability at certain predefined nodes called *Reference Nodes* or *RNs*. Intuitively, the longer a node remains in the neighborhood, the more it broadcasts the *hello* message, and thus it has more chance to be detected. Therefore, the neighbor detection probability at each RN is calculated as a cumulative probability and is an increasing function of time during which a node remains in the neighborhood. More precisely, given time  $t \in [0, \Delta_{neighborhood}]$ , the neighbor detection probability at  $t$  for a RN is equal to the number of all nodes that are detected by RN after being in the neighborhood for time  $t$ , divided by the number of all nodes that are in the neighborhood. Finally, since we use a probabilistic radio propagation model, for each  $t \in [0, \Delta_{neighborhood}]$ , we take the average of the detection probability at  $t$  over all RNs. As the result, we have the values of the average of the detection probability during  $[0, \Delta_{neighborhood}]$  which constitute the output of the simulation.

Note that since we use a probabilistic radio propagation model, two simulations that have the same strategy and environment as the input do not necessarily result in the same output. Thereby, to have a good estimation of the corresponding detection probability of a strategy  $s$  in an environment  $e$ , we take the average of the outputs of five simulations which have  $s$  and  $e$  as the input. We explain this in more detail in Section 3.4.2 after defining the simulation setup.

### 3.4.2 Simulation Setup

We choose the total number of nodes and the location of RNs such that we can estimate the detection probability in the worst case, i.e., where the communication interference and collisions are at the maximum. In fact, if a strategy maximizes the neighbor detection probability in the worst case, we can assume that it maximizes the neighbor detection probability in all cases.

Thus, we consider a square of 100 m width filled with 1000 static nodes located using a uniform random distribution.<sup>2</sup> RNs are the nodes located at the distance

---

<sup>2</sup> In urban environments, a node’s movement may depend on another node’s movement (e.g., if they move in a group). A node can also move on predefined paths or sidewalks [5]. Therefore, topology changes are not always random. However, using a distribution that characterizes such behaviors, requires a particular deployment scenario, i.e., the physical properties of the terrain, the points of interest, etc. Thus, we believe that a random distribution gives a good generic basis for the evaluation.

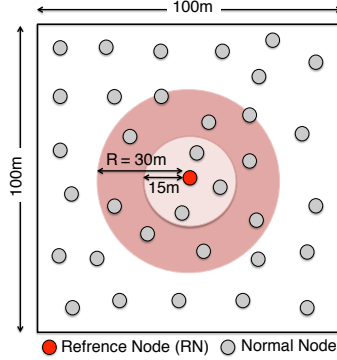


Fig. 3.1: Simulation Map

less than or equal to 5 m from the center of the square. The reason is that the nodes that are close to the center, usually experience the maximum radio interference. The total number of nodes is chosen after studying the *occupancy load factors* of urban surfaces [16]. In architecture and urbanism, the occupancy load factor of a given urban surface defines the maximum number of persons which can occupy one unit area of that surface. The occupancy load factor of an urban surface is mainly defined based on its usage (e.g., residential, office, public assembly, etc...). Although, the occupancy load factors of the urban environments that we consider in this paper are slightly different from one another, 1000 seems to be a good approximation of the maximum number of persons that usually occupy these environments.

For the radio propagation model, we use the implementation of LNS model in ns-2. In a simulation, all nodes have the same idealized transmission range<sup>3</sup> since they are all initialized with a given value of  $pow_{tx}$ . Moreover, even with our lowest  $pow_{tx}$  choice, the idealized transmission range is large enough so that all nodes are within the idealized transmission range of each other. Each RN has the detection range of 30 m. This value is chosen such that even using our lowest  $pow_{tx}$  choice, the detection range is less than the idealized transmission range. Fig. 3.1 depicts the simulation map with only one RN.

For the implementation of 802.11a in ns-2, we use the implementation performed by a team from Mercedes-Benz Research and Development North America and University of Karlsruhe [15]. This implementation includes a completely revised and

<sup>3</sup> The *idealized transmission range* corresponds to the deterministic transmission range calculated for idealized deterministic channel conditions i.e., with no node movement and no obstacles between the sender and the receiver(s).

enhanced architecture for physical and MAC layers to improve the drawbacks of the 802.11 default support in ns-2. In particular, this implementation for the physical layer comprises cumulative *received signal power over noise* (SINR) computation. Its mac layer also accurately implements the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) mechanism.

Thus, we use the default values of this implementation for physical and mac layers [15], however, we disable both *preamble* and *frame body* capture features.<sup>4</sup> For data rate, we consider 6 Mbps using *Binary Phase-Shift Keying* (*BPSK*) modulation scheme and 1/2 coding rate. In fact, more advanced schemes imply higher data rates but also require better received signal quality which reduces the number of receivable packets in the case of 802.11 broadcasts where no acknowledgment or RTS/CTS (Request to Send/Clear to Send) mechanism exist to cope with interferences and collisions.

Based on our assumptions in Section 3.3, the size of a *hello* packet is set to 500 bytes. If messages are generated while previously generated messages are not yet transmitted, the new messages are stored in an interface queue that is capable of storing up to 100 packets. This queue size is chosen such that regardless of the used strategy, no message is dropped by the queue.

To obtain the detection probability for a strategy  $s$  in an environment  $e$ , we perform five simulations with  $s$  and  $e$  as the input and with five different pairs of seeds. In fact, in each simulation one seed is used to initialize the random number generator of the LNS model and the other is used to initialize the random number generator responsible for the randomness of topologies. Thereby, for each simulation, we have a different topology (with different RNs) and a LNS model which is seeded differently. Then, we take the average of the five simulations' outputs. The result is considered as the detection probability for the strategy  $s$  in the environment  $e$ . Recall that the output of a simulation is the average (over all RNs in that simulation) of the neighbor detection probability at time  $t$  for all  $t \in [0, 4]$  seconds. Therefore,

---

<sup>4</sup> *Capture* feature, which is present in some WNICs, can mitigate the effect of collisions to some degrees. Roughly speaking, when a packet collision happens, the capture feature enables the receiver to capture one of the collided packets if certain conditions are fulfilled. The existing WNICs differ in the extent of supporting capturing techniques [29; 15]. There exist two variants of the capture feature (i.e., the *preamble* and the *frame body* capture) in the simulator that we use. However, according to our preliminary tests, enabling these features only increases the chance of packet reception with the same percentage across different strategies and thus, does not influence our conclusions.

the detection probability for the strategy  $s$  in the environment  $e$  is the average (over all RNs in the five simulations) of the neighbor detection probability at time  $t$  for all  $t \in [0, 4]$  seconds.

Note that according to our preliminary tests, with five simulations we already achieve an average that accurately presents the detection probability in each setting. In fact, with five simulations the resulting standard deviation in all cases is very low (in order of  $10^{-2}$ ).

### 3.4.3 Results

According to our simulation results, a node which is situated at a maximum distance of 15 m from a RN, is detected with a high probability (at least 0.8) in all environments. The reason is that at close distances (i.e., up to 15 m), in all environments, the signal strength of a received packet is usually high enough to resist interferences. Therefore, in this section we only discuss the detection of the nodes situated at a distance between 15 m to 30 m from a RN (see Fig. 3.1 on page 67).

Fig. 3.2 on page 71, depicts, as an example, the results for Strategy (15 dBm, 1/4 Sec) in different environments. As shown in the figure and already described in Section 3.4.1, the neighbor detection probability is an increasing function of time. We also observe that for the same strategy, the detection probability increases as we change the environment from the *indoor with hard partitions* to *indoor with soft partitions* and then to *outdoor urban*. This is because the radio signals are attenuated the most in *indoor with hard partitions* and the least in *outdoor urban*.

The fact that the neighbor detection probability is calculated as a cumulative probability and is an increasing function of time enables us to only take into account its last value (i.e., the value at second 4) when comparing the effectiveness of different strategies. Thereby, for the sake of simplicity, we use henceforth the term *neighbor detection probability* while referring to the *neighbor detection probability at second 4*.

Intuitively, in a given environment increasing  $pow_{tx}$  and decreasing  $\Delta_{period}$  should lead to the most effective strategy. In fact, by increasing  $pow_{tx}$ , the packets are transmitted with a more powerful signal and accordingly they better survive the interferences and environmental attenuations. Also, decreasing  $\Delta_{period}$  increments the total number of sent *hellos* and thus increases the chance of reception. However,

simulations show that this is not always true i.e., changing the values of  $pow_{tx}$  and  $\Delta_{period}$ , will not always affect the detection probability in all environments in the same way. However, in certain cases we observe similar behaviors for some range of values, e.g., increasing  $pow_{tx}$  for a given  $\Delta_{period}$  seems to increase the detection probability for the majority of cases (see Fig. 3.4 on page 71), whereas decreasing  $\Delta_{period}$  for a given  $pow_{tx}$  might result in unpredictable behaviors (see Fig. 3.5 on page 71). To understand the reason behind these similarities and differences, we study the mechanism of packet drops by 802.11 physical layer. Based on our study, we define two metrics that can be used to interpret the results in each environment. In the following, we first present the metrics and we show, as an example, how they can be used to interpret the results in one particular environment, i.e., *indoor with hard partitions*. Then, based on the interpretation of the results of different environments using our metrics, we present our general observations on how a change in  $pow_{tx}$  or  $\Delta_{period}$  can affect the detection probability.

### 3.4.3.1 Packet Dropping Metrics

Before defining our metrics, we present an overview of the packet dropping mechanism by 802.11 physical layer. When a packet arrives from the channel to the physical layer of the receiver, its *received signal power over noise* (SINR) is compared to a constant threshold called *SINR threshold*.<sup>5</sup> The threshold value depends on the modulation scheme and the coding rate.<sup>6</sup> If there is only one sender, the noise is equal to the noise floor, which is the sum of the thermal noise of the system plus some additional noise caused by losses in the receiver hardware (e.g. in the antenna cables or electronic parts) [44; 15]. However, if there are other senders which send at the same time, their packets could be sensed by the receiver and increase the noise. If

---

<sup>5</sup> The packet dropping mechanism described in this section is drawn from the *SINR-based reception model* which is adopted by many network simulators (including the simulator that we use). In this model the SINR threshold values are obtained by experimental measurements using real hardware. For more information about this model see [7; 15].

<sup>6</sup> Different modulation schemes and coding rates can be used while transmitting the PLCP (Physical Layer Convergence Procedure) header and the data frame of a 802.11a physical layer packet. Accordingly, the SINR threshold used at the reception can be different for the PLCP header and the data frame. However, since in our simulations we use the same modulation scheme (i.e., BPSK) and coding rate (i.e., 1/2) for both the PLCP header and the data frame, in our description we only consider one SINR threshold.



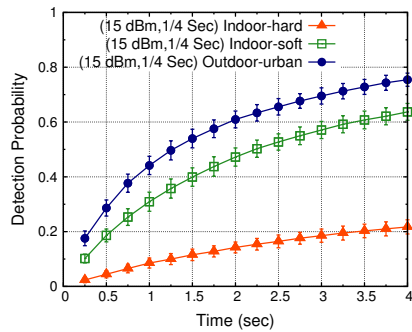


Fig. 3.2: Same strategy in different environments. The vertical error bars present the standard deviation for the detection probability

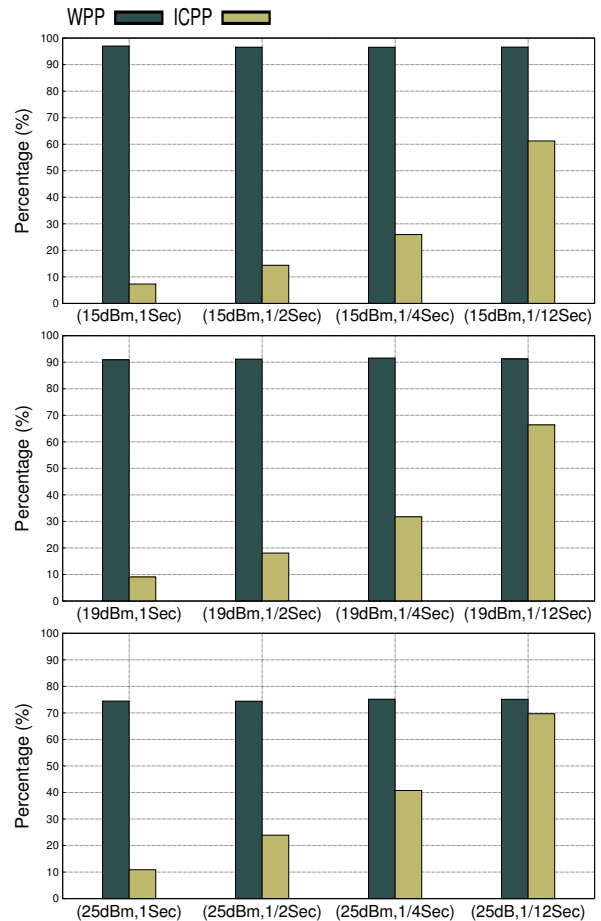


Fig. 3.3: Values of packet dropping metrics for different strategies in the indoor with hard partitions environment

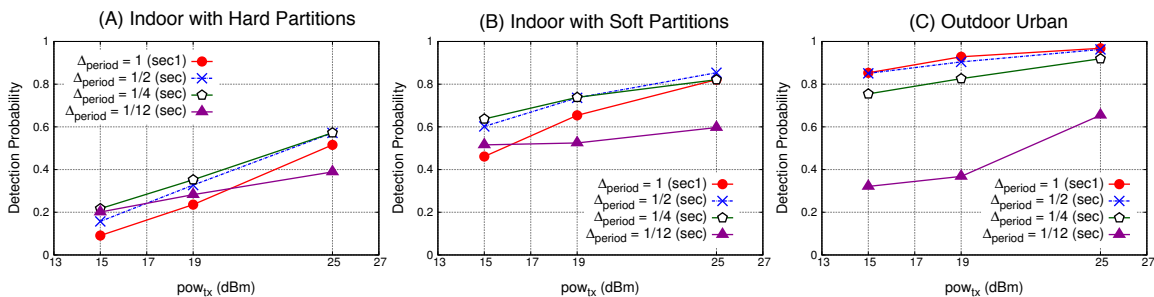


Fig. 3.4: Impact of increasing  $pow_{tx}$  on the detection probability in different environments

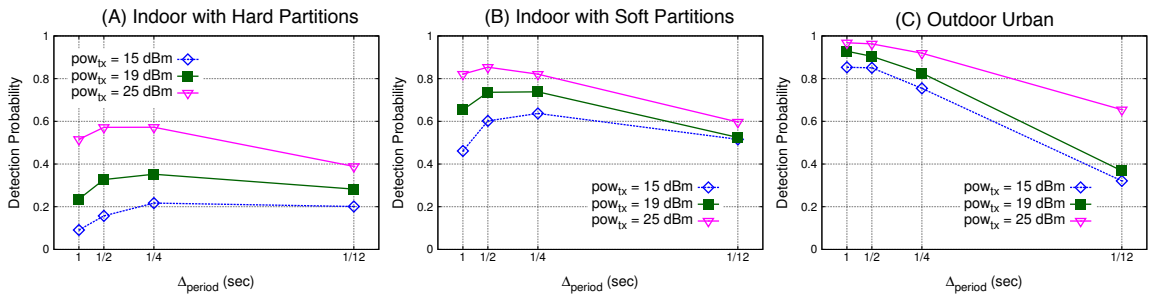


Fig. 3.5: Impact of decreasing  $\Delta_{period}$  on the detection probability in different environments

the SINR of a packet is less than the threshold, no reception process is triggered and the packet is dropped. A packet could also be dropped due to collisions. In this case, a packet is in the reception process, but another packet arrives. If the second packet is strong enough to corrupt the first packet by augmenting the background noise, both the first and the second packets are dropped, otherwise the second packet is dropped and the first packet reception continues. Thus, to explain our simulation results we define two following metrics.

- *Weak Packets Percentage (WPP)*. This is the average percentage of all *weak* packets that arrive to a RN's physical layer out of all sent packets by nodes in a distance of 15 m to 30 m through a simulation. By weak packets, we mean packets which have a low power when they arrive to the physical layer such that even without any interference from other nodes, their SINR is lower than the SINR threshold.
- *Interfered or Collided Packets Percentage (ICPP)*. This is the average percentage of all interfered or collided packets out of all sent packets by nodes in a distance of 15 m to 30 m to a RN through a simulation. By interfered packets, we mean all packets which have an acceptable SINR for reception if there is no interference but their SINR is lower than the SINR threshold because of the interference of other nodes. By collided packets, we mean all packets which are dropped due to collision.

WPP and ICPP are not disjoint i.e., there are packets which are weak but have collided with the reception of another packet. WPP is a function of  $pow_{tx}$  and the environment attenuation (characterized by LNS parameters). ICPP is a function of  $\Delta_{period}$ ,  $pow_{tx}$  and the environment attenuation. A sent packet could also be dropped if it arrives when the receiver's physical layer is in transmission state. However, according to our preliminary evaluations, the percentage of such packets is very low, so we simply ignore them. In addition, in our preliminary experiments, we defined other metrics which are not related to the packet dropping, e.g., average backoff time at senders, however WPP and ICPP seem to interpret the results in a more clear way.

### 3.4.3.2 Metrics-based Interpretation of the Results

We now interpret the results for *indoor with hard partitions* environment using our defined metrics. By using this example, we show how these metrics can help us

to understand the behavior of the detection probability under different strategies. Our discussion is based on the measurements depicted in Fig. 3.3, Fig. 3.4–A and Fig. 3.5–A (all the figures can be found on page 71). In particular, in Fig. 3.3, the values of the defined metrics for different strategies in *indoor with hard partitions* environment are presented.

- *Increasing  $pow_{tx}$  for a given  $\Delta_{period}$ .* As shown in Fig. 3.3, as we increase  $pow_{tx}$  from 15 dBm to 19 dBm and then to 25 dBm, the value of WPP decreases, which means that the percentage of weak (non-receivable) packets that arrive to the physical layer of the receiver decreases. On the other hand, as we increase  $pow_{tx}$ , for the same  $\Delta_{period}$ , the value of ICPP increases. The reason is that increasing  $pow_{tx}$  results in more powerful packets arriving to the physical layer, which can interfere or collide with other packets' reception. So, we observe that when we increase  $pow_{tx}$  considerably, i.e., from 15 dBm or 19 dBm to 25 dBm (recall that dBm is a logarithmic scale), the detection probability increases regardless of the value of  $\Delta_{period}$  (see Fig. 3.4–A). However, when we increase  $pow_{tx}$  from 15dBm to 19dBm, the detection probability improves differently under different values of  $\Delta_{period}$ . For instance, as shown in Fig. 3.4–A, when  $\Delta_{period} = 1/12$  second, increasing  $pow_{tx}$  from 15 dBm to 19 dBm does not improve the detection probability as much as it improves under  $\Delta_{period} = 1$  second. The reason is that under small values of  $\Delta_{period}$ , the value of ICPP is high i.e., many packets are dropped because of collisions and interferences and therefore a small increase in  $pow_{tx}$  cannot improve the detection probability significantly.
- *Decreasing  $\Delta_{period}$  for a given  $pow_{tx}$ .* As depicted in Fig. 3.3, the value of WPP remains the same when decreasing  $\Delta_{period}$ . This is not surprising since WPP is a function of  $pow_{tx}$  and the environment attenuation and is independent from  $\Delta_{period}$ . On the other hand, when decreasing  $\Delta_{period}$ , the value of ICPP increases since more packets arrive per second to the physical layer of the receiver, which increases the chance of collisions and interferences. However, collisions do not have the same impact in the presence of different values of WPP. For instance, as shown in Fig. 3.3, when  $pow_{tx} = 15$  dBm the value of WPP = 96%. In this case, even if the number of collisions increases, a large number of collided packets will be weak (non-receivable) packets. Therefore, decreasing  $\Delta_{period}$  increments the reception chance of the powerful packets. As depicted in Fig. 3.5–A, with  $pow_{tx} = 15$  dBm, the detection probability at  $\Delta_{period} = 1/12$  second is greater than the detection probability at  $\Delta_{period} = 1$  second and almost equal to the detection probability at

$\Delta_{period} = 1/4$  second. However, when the value of WPP is relatively low, collisions have a more significant impact and can decrease the detection probability e.g., as shown in Fig. 3.3, when  $pow_{tx} = 25$  dBm, the value of WPP=75%. In this case, the detection probability at  $\Delta_{period} = 1/12$  second is even less than the detection probability at  $\Delta_{period} = 1$  second (see Fig. 3.5–A).

### 3.4.3.3 Impact of Changing $pow_{tx}$ and $\Delta_{period}$ on Neighbor Detection Probability

After interpreting all results by using the packet dropping metrics, we reach the following general observations regarding the impact of increasing  $pow_{tx}$  and decreasing  $\Delta_{period}$  on the detection probability.

- *Increasing  $pow_{tx}$  for a given  $\Delta_{period}$ .* For a fixed  $\Delta_{period}$ , increasing  $pow_{tx}$  considerably, i.e., from 15 dBm or 19 dBm to 25 dBm, increases the detection probability in all environments (see Fig. 3.4 on page 71). Increasing  $pow_{tx}$  from 15 dBm to 19 dBm, improves the neighbor detection under high values of  $\Delta_{period}$  (e.g., for 1 second), but under low values of  $\Delta_{period}$  (e.g., for 1/12 second), it has less effect and can even lead to no improvement. For instance, as shown in Fig. 3.4–B, in *indoor with soft partitions* environment, under  $\Delta_{period} = 1$  second, increasing power from 15 dBm to 19 dBm increases the detection probability from 0.4612 to 0.6539 (i.e., about 41.78% increase), whereas under  $\Delta_{period} = 1/12$  second, increasing  $pow_{tx}$  from 15 dBm to 19 dBm has almost no influence on the detection probability. This is because under low values of  $\Delta_{period}$ , the number of collisions and interferences is relatively high.

Furthermore, although in all environments, increasing  $pow_{tx}$  improves the detection probability, in general, the improvement becomes less significant as we move from a more obstructed environment to a less obstructed environment. For instance, under  $\Delta_{period} = 1$  second, increasing  $pow_{tx}$  from 15 dBm to 25 dBm in the *indoor with hard partitions* environment increases the detection probability by a factor of 5.66 (see Fig. 3.4–A), whereas, in the *outdoor-urban* environment, it increases the detection probability by a factor of 1.13 (see Fig. 3.4–C). In fact, as the environment becomes less obstructed, packets do not need to be transmitted with a very powerful signal to resist the environmental attenuation, thus, increasing  $pow_{tx}$  results in a less significant improvement of the detection probability.

- *Decreasing  $\Delta_{period}$  for a given  $pow_{tx}$ .* For a fixed value of  $pow_{tx}$ , decreasing  $\Delta_{period}$  could have different impacts on the detection probability depending on the environment (see Fig. 3.5 on page 71). For instance, in indoor environments, decreasing  $\Delta_{period}$  down to a certain value (e.g., 1/4 second in *indoor with hard partitions*) can increase the detection probability. However, below this value the detection probability starts to decrease due to the increase in collisions and interferences. In *outdoor urban*, decreasing  $\Delta_{period}$  generally decreases the detection probability. This is because the packets are less attenuated by the environment (compared to indoor environments) and a good percentage of them arrive to the receiver's physical layer with acceptable SINR. Thus, decreasing  $\Delta_{period}$  only increases collisions and prevents the reception of the acceptable packets.

#### 3.4.3.4 The Most Effective Strategy

Fig. 3.6 on page 76, depicts the strategies ranked in descending order with respect to their resulting detection probability in different environments. As shown, the most effective strategy is not the same in all environments. More precisely, Strategy (25 dBm, 1/4 Sec), Strategy (25 dBm, 1/2 Sec) and Strategy (25 dBm, 1 Sec) are respectively the most effective strategies in *indoor with hard partitions*, *indoor with soft partitions* and *outdoor urban areas*. These results are justified by our observations in Section 3.4.3.3.

Based on the detection probability comparisons, for a given strategy  $s$  and environment  $e$ , we defined the following metrics:

- *Effectiveness rank.* This rank is out of 12 strategies in set  $S$  and is based on the ranking in Fig. 3.6, i.e., in descending order with respect to detection probability. The effectiveness rank of the most effective strategy is 1.
- *Effectiveness ratio.* This is the ratio of the detection probability of  $s$  to the detection probability of the most effective strategy in environment  $e$ . Informally speaking, it compares the effectiveness of  $s$  to the maximum effectiveness that can be achieved in environment  $e$ . The effectiveness ratio of the most effective strategy is 1.

Table 3.2 on page 76, depicts the effectiveness ranks and ratios of the strategies in different environments. We use these metrics later in Section 3.6 when discussing the tradeoff between effectiveness and efficiency.

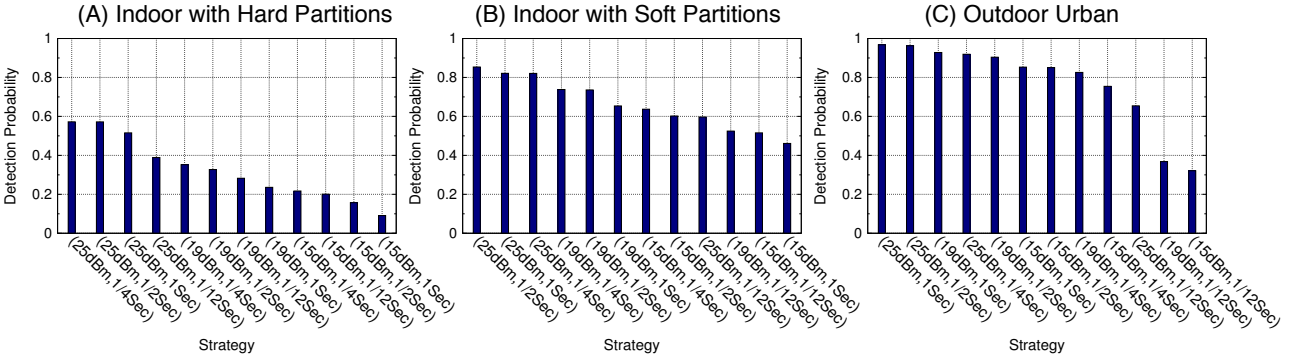


Fig. 3.6: Strategies ranked in the descending order with respect to their corresponding detection probability in different environments

Table 3.2: Effectiveness rank and ratio of the strategies in different environments

	Strategy												
	(15 dBm, 1 Sec)	(15 dBm, 1/2 Sec)	(15 dBm, 1/4 Sec)	(15 dBm, 1/12 Sec)	(19 dBm, 1 Sec)	(19 dBm, 1/2 Sec)	(19 dBm, 1/4 Sec)	(19 dBm, 1/12 Sec)	(25 dBm, 1 Sec)	(25 dBm, 1/2 Sec)	(25 dBm, 1/4 Sec)	(25 dBm, 1/12 Sec)	
Indoor-hard	Rank	12	11	9	10	8	6	5	7	3	2	1	4
	Ratio	0.16	0.27	0.37	0.35	0.41	0.57	0.61	0.49	0.90	0.99	1	0.67
Indoor-soft	Rank	12	8	7	11	6	5	4	10	2	1	3	9
	Ratio	0.54	0.70	0.74	0.60	0.76	0.862	0.864	0.61	0.962	1	0.961	0.69
Outdoor-urban	Rank	6	7	9	12	3	5	8	11	1	2	4	10
	Ratio	0.88	0.87	0.77	0.33	0.95	0.93	0.85	0.38	1	0.99	0.94	0.67

### 3.5 Evaluation of Efficiency

Efficiency is measured by the inverse of energy consumption per process. Therefore, in this section we first define a model of energy consumption and then design an algorithm that, based on the model, calculates for each strategy the energy consumption per process. After describing the algorithm, we present the results and we

discuss how a change in  $pow_{tx}$  or  $\Delta_{period}$  can affect the energy consumption. Finally we compare the efficiency of the strategies and present the most efficient strategy.

### 3.5.1 Energy Consumption Model

Communication is the primary cause of energy consumption of a node executing the neighbor detection algorithm. Since 802.11 (or WiFi) communication is considered one of the main causes of the battery discharge in mobile devices [33; 30], in this section we only consider the energy consumption of the 802.11a wireless network interface card (or WNIC). As already described in Section 3.3, WNIC is the component that implements the MAC and the physical layers of the OSI model in a mobile device. To calculate the energy consumption of the WNIC for each strategy, we first define its energy consumption model as below.

Power is defined as the amount of energy consumed per unit of time. It is known that a 802.11 WNIC exhibits different power consumptions at different radio modes. Therefore, in order to define the energy consumption model of the WNIC, we should first identify its different radio modes.

In this paper, we assume that the WNIC does not use any power saving mechanism for the IEEE 802.11 distributed coordination function (DCF). In fact, according to the power saving mechanism of the 802.11 standard, the WNIC sleeps most of the time and wakes up periodically to check whether there are some packets that it should transmit or receive. During the sleep mode no transmission or reception is possible and the power consumption is extremely low. Thus, If a node has a packet to transmit, it should buffer the packet and waits until the next wake-up time [1]. In this paper, we do not use the power saving mechanism for two main reasons. First, waking up at the right moment requires the nodes to have synchronized clocks. Second, the power saving mechanism is known to perform poorly when the number of nodes is high. In fact, at the wake-up time, a sender node should first announce the list of the buffered packets to destinations. The announcement is performed by sending an *ad-hoc traffic indication map (ATIM)* packet. As the number of nodes increases, more ATIM transmissions could take place at the same time which result in more collisions and hence lower performance [43]. Although some researchers proposed new power saving mechanisms without some of these limitations (for instance, schemes that work with asynchronous clocks [25; 55]), in this paper for simplicity's sake we do

not use any of these mechanisms. We might consider their potential use in our future work.

Thus, we assume that the WNIC can only operate in one of three radio modes, namely, *transmit*, *receive* and *idle* [1]. As their names suggest, the transmit and the receive modes correspond respectively to the cases where the WNIC transmits or receives a packet. In general, power consumption in the transmit mode is different from power consumption in the receive mode, since different circuits are used in these modes [38]. In the idle mode, the WNIC is required to continuously sense the medium. Thus, intuitively the power consumption in the idle mode should be similar to the power consumption in the receive mode. The experimental results in [19] confirm this fact and show that the power consumption in the idle mode is only slightly different from the power consumption in the receive mode. Therefore, we assume that in the idle mode, the WNIC consumes the same amount of power as in the receive mode. This assumption results in two general radio modes:

- *Active mode*. This mode is characterized by power consumption  $pow_{active}$  and corresponds to the cases where WNIC is in the transmit mode.
- *Passive mode*. This mode is characterized by power consumption  $pow_{passive}$  and corresponds to the cases where the WNIC is either in the idle or the receive modes.

Knowing these two general radio modes, the energy consumption model can be specified as follows. Let  $T_{total}$  be the time duration for which the energy consumption of WNIC is defined. Then,  $T_{total}$  can be split into  $T_{active}$  and  $T_{passive}$ , which denote respectively the duration spent in the active and passive modes. Since power consumption of each mode is known, the energy consumption of each mode can be calculated separately. Let  $E_{total}$  denote the total energy consumption of WNIC during  $T_{total}$ , then  $E_{total}$  can be found by summing  $E_{passive}$  and  $E_{active}$ , where  $E_{passive}$  and  $E_{active}$  denote respectively the energy consumption in the active and passive modes [14].

### 3.5.2 Energy Consumption Calculation Algorithm

To obtain the energy consumption of the WNIC for different strategies, we devise an algorithm called *energy consumption calculation* algorithm. It is based on the energy consumption model described in Section 3.5.1. It calculates the active, passive and



total energy consumption of the WNIC for a given strategy and for a given time duration. The algorithm calculates the energy consumption independently of the environment. The main reason is that the time that the WNIC of a node (executing the neighbor detection algorithm) spends in each radio mode is independent of the environment attenuation.

Moreover, in order for the algorithm to characterize accurately the energy consumption of a current smartphone's WNIC, we use the power consumption specifications of Broadcom's BCM 4328 WNIC which is also used as a reference in [39] to devise power consumption equations.

The algorithm has three input parameters: the transmission power  $pow_{tx}$ , the time duration  $\Delta_{period}$  (these two parameters form Strategy  $(pow_{tx}, \Delta_{period})$ ), and the time duration  $T_{total}$  for which the energy consumption is calculated. The algorithm has three output parameters: the energy consumption in the active mode  $E_{active}$ , the energy consumption in the passive mode  $E_{passive}$  and the total energy consumption  $E_{total}$ .

The algorithm has two main steps:

1. For  $m \in \{active, passive\}$ , perform the following:
  - a. Calculate  $pow_m$ , where  $pow_m$  is the power consumption of  $m$  mode.
  - b. Calculate  $T_m$ , where  $T_m$  is the amount of time out of  $T_{total}$  that is spent in  $m$  mode.
  - c. Set  $E_m = pow_m \times T_m$ , where  $E_m$  is the amount of energy consumed in  $m$  mode.
2. Set  $E_{total} = \sum_{m \in \{active, passive\}} E_m$ , where  $E_{total}$  is the total energy consumed during  $T_{total}$  and return  $E_{active}$ ,  $E_{passive}$  and  $E_{total}$  as output.

In the following, we describe two Substeps 1a and 1b.

1a) Calculate  $pow_m$  : depending on the value of  $m$ , there exist two cases:

- If  $m$  is equal to *active*,  $pow_{active}$  should be calculated. To do so, we use Equation 3.1 where all quantities are in milliwatts (mWs). This equation was introduced in [39]. As discussed in Section 3.5.1,  $pow_{active}$  is the power consumed by WNIC while transmitting packets. Equation 3.1 shows how  $pow_{active}$  can be defined in terms of transmission power  $pow_{tx}$ .

$$pow_{active} = 305 + \frac{pow_{tx}}{0.02 \times 5^{\lceil \frac{2}{3} \times \log_{10}(pow_{tx}) \rceil}} \quad (3.1)$$

The first term of the equation, i.e., 305 mW, represents the common power consumed by the circuitry independent of  $pow_{tx}$ . This value is obtained based on Broadcom's BCM 4328 WNIC specifications [13]. The second term of the equation represents the total power consumed by the RF power amplifier of the WNIC, which is active during transmissions.

- If  $m$  is equal to *passive*,  $pow_{passive}$  should be calculated. In [39], power consumed by WNIC during the reception is assumed to be always equal to 295 mW based on Broadcom's BCM 4328 WNIC specifications. This assumption seems to be correct since the experimental results in [18] show that the amount of power consumed by the WNIC during the reception is not influenced by the transmission power  $pow_{tx}$ . Thus, we also adopt this assumption in our algorithm and set the value of  $pow_{passive}$  to 295 mW.

1b) Calculate  $T_m$  : depending on the value of  $m$ , there exist two cases:

- If  $m$  is equal to *active*,  $T_{active}$  should be calculated. Intuitively,  $T_{active}$  is the sum of transmission times of all packets that are transmitted by WNIC during  $T_{total}$ . Thus, let  $T_{tx}$  be the transmission time of a packet and  $n$  be the number of packets transmitted during  $T_{total}$ ,  $T_{active}$  can be defined as:

$$T_{active} = n \times T_{tx} = \left\lfloor \frac{T_{total}}{\Delta_{period}} \right\rfloor \times 0.000728 \quad (3.2)$$

where all quantities are in seconds. Note that in Equation 3.2,  $n$  and  $T_{tx}$  are respectively replaced by  $\left\lfloor \frac{T_{total}}{\Delta_{period}} \right\rfloor$  and 0.000728. Below we explain how these values are obtained.

We find the value of  $T_{tx}$  using Equation 3.3. This Equation was introduced in [38]. It calculates the transmission time of an *802.11a* data frame in seconds

given its payload size  $\mathcal{L}$  in bytes and the Bytes-per-Symbol information ( $\mathcal{BpS}$ ) which is itself a function of data rate  $\mathcal{R}$ .

$$T_{tx} = 0.00002 + \left\lceil \frac{30.75 + \mathcal{L}}{\mathcal{BpS}(\mathcal{R})} \right\rceil \times 0.000004 \quad (3.3)$$

In our case, i.e., with data rate of 6 Mbps,  $\mathcal{BpS} = 3$ . Since  $\mathcal{L} = 500$  bytes, we find  $T_{tx} = 0.000728$  seconds.

To find the value of  $n$ , we should find the number of messages transmitted during  $T_{total}$ . According to the neighbor detection algorithm, a message is sent by the application layer every  $\Delta_{period}$ . So, by setting the value of  $n$  to  $\left\lceil \frac{T_{total}}{\Delta_{period}} \right\rceil$ , we make three assumptions. First, we assume that one application layer message results in one MAC frame. This assumption conforms to our simulation study in Section 3.4.<sup>7</sup> Second, we assume that no message is dropped by the interface queue. This assumption conforms to our system model. Third, we assume that the amount of time spent by a message between its sending by the application layer until the end of its transmission from the physical layer is smaller than  $\Delta_{period}$ . This assumption is also reasonable considering the values of  $\Delta_{period}$  of the studied strategies and is also confirmed by our simulation study described in Section 3.4. Based on these assumptions, we know that a message sent by the application layer is always transmitted to the channel before the sending of the next message by the application layer. Hence, all messages sent by the application layer during  $T_{total}$  are transmitted during  $T_{total} + \epsilon$ , with  $\epsilon$  being negligible ( $\epsilon$  accounts for limit conditions where the last message is sent very close to the end of the measurement period).

- If  $m$  is equal to *passive*,  $T_{passive}$  should be calculated.  $T_{passive}$  is the part of  $T_{total}$  that the WNIC spends in the reception or the idle modes, i.e., it is the part of  $T_{total}$  that WSN does not spend in the active mode. Therefore,  $T_{passive}$  is calculated using Equation 3.4 where  $T_{active}$  is replaced by Equation 3.2. Note that all quantities in Equation 3.4 are in seconds.

---

<sup>7</sup> In real world applications, this assumption is not always true. However, to prevent a high number of packet collisions and network congestion, the *hello* packets are usually small packets resulting in few MAC frames. In addition,  $T_{active}$  is a linear function of number of transmitted frames. Therefore, this assumption does not influence our observations in Section 3.5.3.

$$\begin{aligned}
T_{passive} &= T_{total} - T_{active} = T_{total} - (n \times T_{tx}) \\
&= T_{total} - \left( \left\lfloor \frac{T_{total}}{\Delta_{period}} \right\rfloor \times 0.000728 \right)
\end{aligned} \tag{3.4}$$

### 3.5.3 Results

Using the energy consumption evaluation algorithm, we calculated  $E_{active}$ ,  $E_{passive}$  and  $E_{total}$  for each strategy during  $T_{total} = \Delta_{neighborhood} = 4$  seconds. Based on these calculations, we reached some observations described below. Note that these observations are valid for any value of  $T_{total}$  since the output energies of the energy consumption algorithm are linear functions of  $T_{total}$ .

- *Regardless of strategy, the majority of  $E_{total}$  consists of  $E_{passive}$ .* The results of different strategies show that, on average, 98.9% of  $E_{total}$  consists of  $E_{passive}$  and only 1.08% consists of  $E_{active}$ . Roughly speaking, the reason is that regardless of the used strategy, the WNIC spends much more time in the passive mode than in the active mode. In fact, among all tested strategies, Strategy (25 dBm, 1/12 Sec) is the one that has the maximum value for  $E_{active}$  and at the same time the minimum value for  $E_{passive}$ .<sup>8</sup> When we compare  $T_{active}$  and  $pow_{active}$  of this strategy with its corresponding  $T_{passive}$  and  $pow_{passive}$ , we realize that its  $pow_{active}$  is 4.69 times greater than  $pow_{passive}$  (recall that  $pow_{passive}$  is always equal to 295 mW). However, its corresponding  $T_{active}$  is still about 113.46 times smaller than its  $T_{passive}$ . That is why its resulting  $E_{active}$  is still much smaller than its  $E_{passive}$ .
- *$E_{active}$  has a high variation between different strategies while  $E_{passive}$  remains more or less constant.* Our results reveal that by changing strategy,  $E_{active}$  varies a lot whereas  $E_{passive}$  tends to remain more or less the same. In fact, let  $E_{active}^{max}$  and  $E_{active}^{min}$  denote respectively the maximum and minimum of  $E_{active}$

---

<sup>8</sup> A strategy can maximize  $E_{active}$  if it can maximize at the same time  $T_{active}$  and  $pow_{active}$ . Roughly speaking, it is the strategy that results in the maximum number of transmitted packets and the maximum transmission power. On the other hand, a strategy can maximize  $E_{passive}$  if it can just maximize  $T_{passive}$ , since  $pow_{passive}$  is constant and not a function of transmission power. In other words, all strategies that result in the minimum number of transmitted packets, maximize  $E_{passive}$ . A similar reasoning can be applied to find the strategies that minimize  $E_{active}$  and  $E_{passive}$ .

for the tested strategies, and  $E_{passive}^{max}$  and  $E_{passive}^{min}$  denote respectively the maximum and minimum of  $E_{passive}$  for the tested strategies. Then,  $E_{active}^{max}$  is 26.7821 times greater than  $E_{active}^{min}$ , whereas  $E_{passive}^{max}$  is more or less equal to  $E_{passive}^{min}$  since  $E_{passive}^{max}/E_{passive}^{min} = 1.0080$  (see Fig. 3.7 on page 84).

- *Variation of  $E_{total}$  between different strategies is mainly due to variation of  $E_{active}$ .* This observation is the direct consequence of the observation described in the previous point. Intuitively, since  $E_{active}$  varies a lot between strategies while  $E_{passive}$  remains more or less constant, the variation of  $E_{total}$  is mainly due to  $E_{active}$ . For instance, among the tested strategies, Strategy (25 dBm, 1/12 sec) is the one that maximizes  $E_{total}$  and Strategy (15 dBm, 1 sec) is the one that minimizes  $E_{total}$ . Thus, if we change the strategy from (25 dBm, 1/12 Sec) to (15 dBm, 1 Sec) we save 3.05% in  $E_{total}$ . This is the result of saving 3.82% in  $E_{active}$  and at the same time losing 0.77% in  $E_{passive}$ .

Based on these observations, we directly consider the active energy consumption instead of the total energy consumption while measuring the efficiency of each strategy. As shown in Fig. 3.7, under  $T_{total} = \Delta_{neighborhood} = 4$  seconds, the amount of energy saved by applying the strategy with minimum  $E_{active}$  instead of other strategies is very low ( i.e., at most 47 milliJoules (mJ) ). However, since  $E_{active}$  is a linear function of time, this amount becomes more and more significant as  $T_{total}$  increases (see Fig. 3.8 on page 84). In other words, choosing a strategy that minimizes the active energy consumption will not result in saving much energy in the short-term but in the long-term. Note that, henceforth, we use the term *energy consumption* instead of *active energy consumption* for simplicity's sake.

### 3.5.3.1 Impact of Changing $pow_{tx}$ and $\Delta_{period}$ on Energy Consumption

In this section, we discuss the impact of increasing  $pow_{tx}$  and decreasing  $\Delta_{period}$  on energy consumption.

- *Increasing  $pow_{tx}$  for a given  $\Delta_{period}$ .* According to the energy consumption calculation algorithm, for a given  $\Delta_{period}$ , when we increase  $pow_{tx}$ , we in fact, increase the power consumption in the active mode or  $pow_{active}$  which in its turn increases the energy consumption. Thus,  $pow_{active}$  is about 621.22 mW, 822.13 mW and 1386.48 mW when transmitting with  $pow_{tx}$  of 15 dBm, 19 dBm and 25 dBm, respectively. This means that at a given  $\Delta_{period}$ , energy consumption is slightly

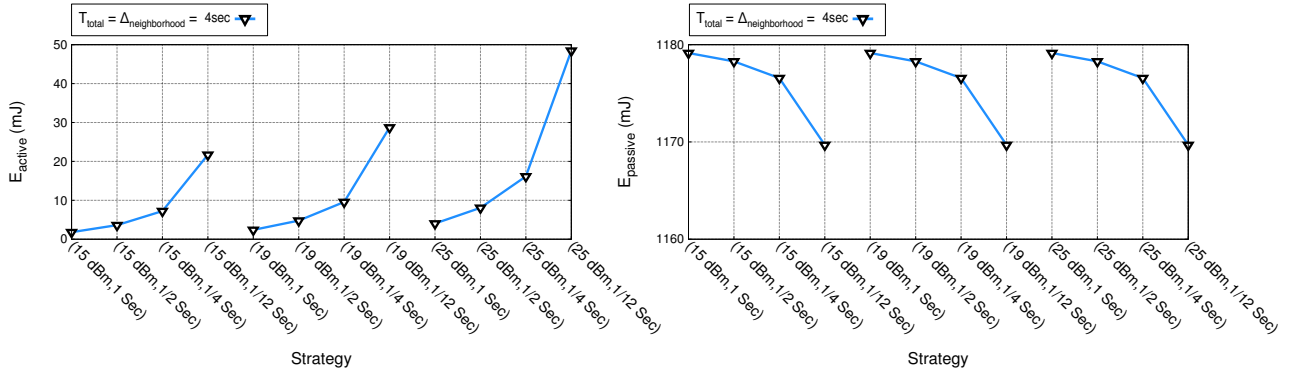


Fig. 3.7: Variation of  $E_{active}$  and  $E_{passive}$  over different strategies when  $T_{total} = \Delta_{neighborhood} = 4$  seconds

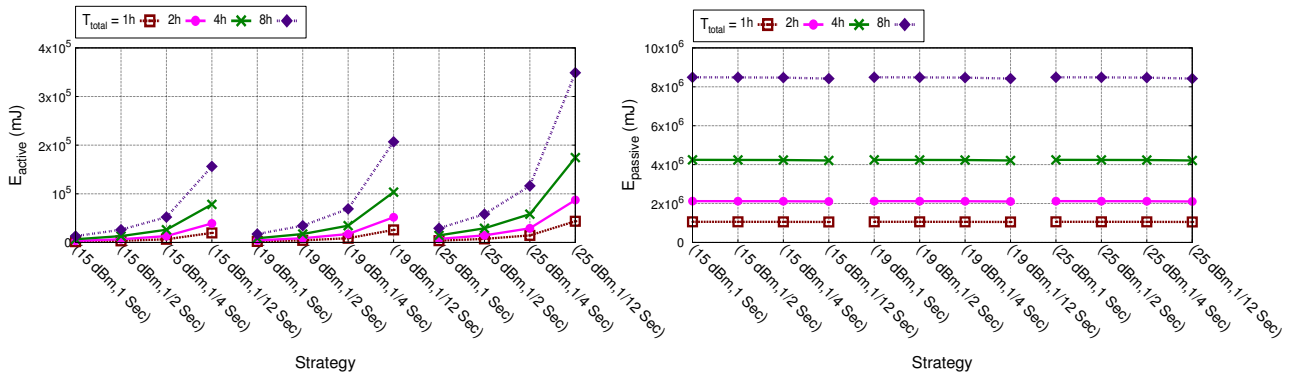


Fig. 3.8: Variation of  $E_{active}$  and  $E_{passive}$  over different strategies for high values of  $T_{total}$  (different hours)

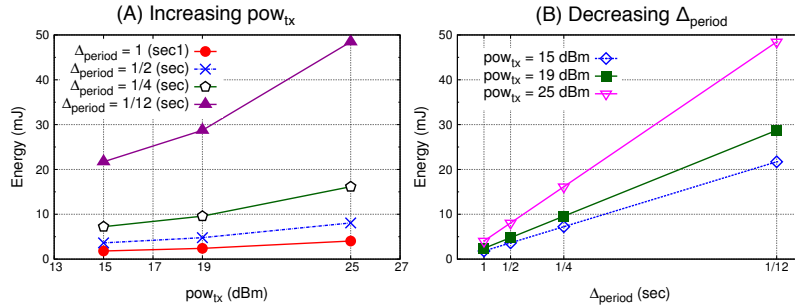


Fig. 3.9: Impact of increasing  $pow_{tx}$  and decreasing  $\Delta_{period}$  on the energy consumption

increased (by 1.3 times) when we increase  $pow_{tx}$  from 15 dBm to 19 dBm and is almost doubled when we increase  $pow_{tx}$  from 15 dBm or 19 dBm to 25 dBm (see Fig. 3.9-A on page 84).

- *Decreasing  $\Delta_{period}$  for a given  $pow_{tx}$ .* According to the energy consumption calculation algorithm,  $E_{active}$  is a linear function of  $1/\Delta_{period}$ . Thus, for a given  $pow_{tx}$ , decreasing  $\Delta_{period}$  increases the energy consumption in a linear manner. This increase is at least twice. Recall that by increasing  $pow_{tx}$ , the energy consumption is at most doubled. Thereby, the impact of decreasing  $\Delta_{period}$  on energy consumption is usually more significant than the impact of increasing  $pow_{tx}$  (see Fig. 3.9-B on page 84).

### 3.5.3.2 The Most Efficient Strategy

Fig. 3.10 on page 86, depicts the strategies ranked in ascending order with respect to their corresponding energy consumption. As discussed in Section 3.5.3.1, compared to increasing  $pow_{tx}$ , decreasing  $\Delta_{period}$  (which accordingly increases the number of transmitted packets) has generally a more significant impact on the increase of energy consumption. That is why for instance, Strategy (15 dBm, 1/2 Sec) consumes more energy than Strategy (19 dBm, 1 Sec) in spite of the fact that Strategy (15 dBm, 1/2 Sec) uses a lower transmission power compared to Strategy (19 dBm, 1 Sec).

As depicted in Fig. 3.10, among all tested strategies, Strategy (15 dBm, 1 Sec) and Strategy (25 dBm, 1/12 Sec) consume the minimum and maximum amount of energy, respectively. Therefore, the most efficient strategy is Strategy (15 dBm, 1 Sec). Note that the most efficient strategy is the same in all environments since energy consumption is calculated independently of environment.

Based on the energy consumption comparisons, for a given strategy  $s$ , we defined the two following metrics:

- *Efficiency rank.* This rank is out of 12 strategies in set  $S$  and is based on the ranking in Fig. 3.10 i.e., in ascending order with respect to energy consumption. Efficiency rank of the most efficient strategy is equal to 1.
- *Efficiency ratio.* This is the ratio of energy consumption of the most efficient strategy to the energy consumption of  $s$ . Informally speaking, it compares the efficiency of  $s$  to the maximum efficiency that can be achieved. The efficiency ratio of the most efficient strategy is 1.

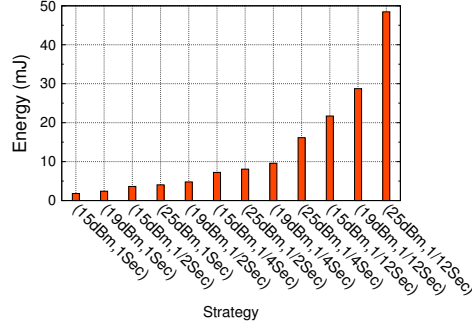


Fig. 3.10: Strategies ranked in the ascending order with respect to their corresponding energy consumption

Table 3.3: Efficiency rank and ratio of the strategies

	Strategy											
	(15 dBm, 1 Sec)	(15 dBm, 1/2 Sec)	(15 dBm, 1/4 Sec)	(15 dBm, 1/12 Sec)	(19 dBm, 1 Sec)	(19 dBm, 1/2 Sec)	(19 dBm, 1/4 Sec)	(19 dBm, 1/12 Sec)	(25 dBm, 1 Sec)	(25 dBm, 1/2 Sec)	(25 dBm, 1/4 Sec)	(25 dBm, 1/12 Sec)
<b>Rank</b>	1	3	6	10	2	5	8	11	4	7	9	12
<b>Ratio</b>	1	0.5	0.25	0.08	0.75	0.37	0.18	0.06	0.44	0.22	0.11	0.03

Table 3.3, depicts the efficiency ranks and ratios of the strategies. We use these metrics later in Section 3.6 when discussing the tradeoff between effectiveness and efficiency.

### 3.6 Effectiveness-Efficiency Tradeoff

In this section, we first compare the results of effectiveness and efficiency evaluations to find the strategy that maximizes both effectiveness and efficiency for each environment. We show that there is a conflict between effectiveness and efficiency. Hence, such a strategy does not exist in any environment. We then propose an approach to make a tradeoff between effectiveness and efficiency and we find the tradeoff strategy for each environment. Finally, to show how good the tradeoff strategy is, for



each environment, we compare its effectiveness and its efficiency to the maximum effectiveness and efficiency that can be achieved in that environment.

### 3.6.1 Conflict

A summary of the results of the effectiveness and efficiency evaluations can be found in Tables 3.4 and 3.5 on page 92.<sup>9</sup> As depicted in Table 3.4, the most effective strategy and the most efficient strategy are not the same in any environment.

In fact, there is a conflict when we try to maximize both effectiveness and efficiency. The reason is that, most of the time, a change in  $pow_{tx}$  or  $\Delta_{period}$  does not influence effectiveness and efficiency similarly and in some cases it results in opposite behaviors (see Table 3.5). For instance, consider  $pow_{tx}$ . Based on the evaluations, we know that regardless of environment, increasing  $pow_{tx}$  increases effectiveness, whereas it decreases efficiency. That is why the most effective strategy has the highest  $pow_{tx}$  (i.e., 25 dBm) in all environments whereas the most efficient strategy has the lowest  $pow_{tx}$  (i.e., 15 dBm). Also, regarding  $\Delta_{period}$ , we know that decreasing  $\Delta_{period}$  (down to some degree) increases effectiveness in indoor environments and decreases effectiveness as the environment becomes less obstructed. At the same time, decreasing  $\Delta_{period}$  decreases efficiency. Thereby,  $\Delta_{period}$  of the most effective strategy increases from 1/4 second to 1/2 second and then to 1 second as we change the environment from *indoor with hard partitions* to *indoor with soft partitions* and then to *outdoor urban areas*, respectively. On the other hand, the most efficient strategy has the highest  $\Delta_{period}$  (i.e., 1 second).

Note however that the conflict becomes less severe as the environment becomes less obstructed. For instance, as we change the environment from the *indoor with hard partitions* to *indoor with soft partitions* and then to *outdoor urban areas*, the efficiency ratio of the most effective strategy increases from 0.11 to 0.22 and then to 0.44, respectively. As a result, its efficiency rank increases from 9th to 7th and then to 4th (see Table 3.4). In fact, as already discussed, the impact that a change in  $\Delta_{period}$  has on effectiveness and efficiency becomes similar as we move from indoor environments to outdoor. Therefore,  $\Delta_{period}$  of the most effective strategy becomes

---

<sup>9</sup> Tables 3.4 and 3.5 summarize the results of different sections of this paper. In this section, we do not discuss the results in rows or columns corresponding to *tradeoff strategy* or *BCR*.

higher (and closer to  $\Delta_{period}$  of the most efficient strategy) in less obstructed environments.

Similarly, the most efficient strategy becomes more effective as the environment becomes less obstructed. More precisely, as depicted in Table 3.4, as we change the environment from the *indoor with hard partitions* to *indoor with soft partitions* and then to *outdoor urban areas*, the effectiveness ratio of the most efficient strategy increases from 0.16 to 0.54 and then to 0.88, respectively. As a result, its effectiveness rank increases from 12th position to 6th position (see Table 3.4). In fact, compared to other strategies, the most efficient strategy has the lowest  $pow_{tx}$  (i.e., 15 dBm) and the highest  $\Delta_{period}$  (i.e., 1 second). As already discussed, the positive impact of a high  $\Delta_{period}$  on effectiveness becomes more significant as the environment becomes less obstructed. At the same time, based on the results in Section 3.4.3.3, we know that the negative impact that a low  $pow_{tx}$  has on effectiveness becomes less considerable as the environment becomes less obstructed.

### 3.6.2 Approach to Make the Tradeoff

Up to this point, our goal was to find, for each environment, the strategy that has both maximum effectiveness and efficiency among all other strategies. In order to achieve our goal, we evaluated the effectiveness and the efficiency of each strategy separately. However, due to the conflict described in Section 3.6.1, such strategy does not exist. Moreover, in a given environment, the strategy that has the highest effectiveness usually has a low efficiency and the strategy that has the highest efficiency is not always very effective. Therefore, we need to find a tradeoff strategy, that is a strategy that on one hand does not consume a lot of energy and on the other hand results in a high (but maybe not the highest) detection probability compared to other strategies.

The main idea for making the tradeoff comes from the concept of *cost–benefit analysis* in the field of economy [9]. Thus, we identify the resulting detection probability of a strategy as its benefit and its resulting energy consumption as its cost. Then, for each strategy, the *benefit–cost ratio (BCR)* is calculated. Finally, the strategy that results in the highest ratio is chosen as the tradeoff strategy. This means that the tradeoff strategy is the one that makes the best use of energy for neighbor detection.

### 3.6.3 Benefit–Cost Ratio (BCR)

Let  $e$  be the environment in which we apply a strategy  $s$ . Also, let  $Pr_{detect}(s, e)$  denote the neighbor detection probability obtained by applying  $s$  in  $e$  and  $E(s)$  denote the energy consumption due to applying  $s$ . Then, BCR achieved by applying  $s$  in  $e$  or  $BCR(s, e)$  is defined as:

$$BCR(s, e) = \frac{Pr_{detect}(s, e)}{E(s)} \quad (3.5)$$

### 3.6.4 Impact of changing $pow_{tx}$ and $\Delta_{period}$ on BCR

We use Fig. 3.11 and Fig. 3.12 on page 90 to discuss the impact of changing  $pow_{tx}$  and  $\Delta_{period}$  on BCR. Intuitively, changing the value of  $pow_{tx}$  or  $\Delta_{period}$  can only improve BCR if the resulting gain in the detection probability is more significant than the loss of energy. More precisely, suppose that a change in the value of one of these parameters, changes the detection probability by a factor of  $x$  and the energy consumption by a factor of  $y$ . Then, BCR is increased if  $x/y > 1$ . If  $x/y = 1$ , BCR does not change. Finally, if  $x/y < 1$ , BCR decreases. In the following, we use this observation to interpret the results.

- *Increasing  $pow_{tx}$  for a given  $\Delta_{period}$ .* We know that for a fixed value of  $\Delta_{period}$ , increasing  $pow_{tx}$  increases the detection probability in all environments. It also increases the energy consumption. However as shown in Fig. 3.11, it can have different impacts on the value of BCR depending on the environment. For instance, under  $\Delta_{period} = 1$  second, increasing  $pow_{tx}$  from 15 dBm to 25 dBm increases BCR in the *indoor with hard partitions* environment and decreases BCR in the *outdoor-urban* environment. In fact, in both cases the energy consumption is increased by a factor of 2.23 (recall that the energy consumption is independent of environment). However, in the *indoor with hard partitions* environment the detection probability is increased by a factor of 5.66, whereas, in the *outdoor urban* environment, it is only increased by a factor of 1.13.

To summarize the results we can say that increasing  $pow_{tx}$  improves BCR only in indoor environments, under high values of  $\Delta_{period}$  and for certain ranges of  $pow_{tx}$  (see Fig. 3.11). In fact, as already discussed in Section 3.4.3.3, although in all environments increasing  $pow_{tx}$  improves the detection probability, in general, the

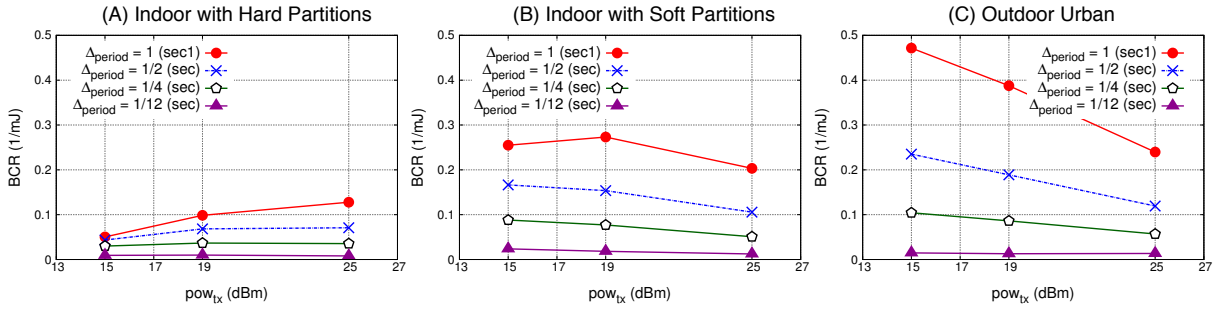


Fig. 3.11: Impact of increasing  $pow_{tx}$  on BCR in different environments

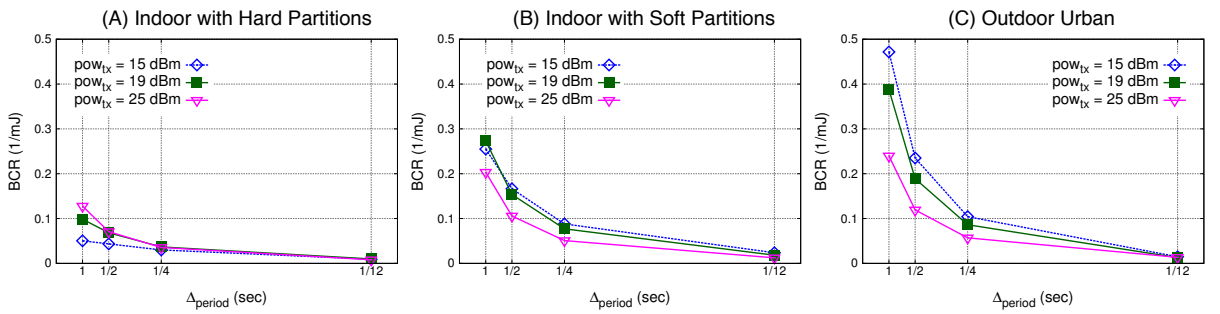


Fig. 3.12: Impact of decreasing  $\Delta_{period}$  on BCR in different environments

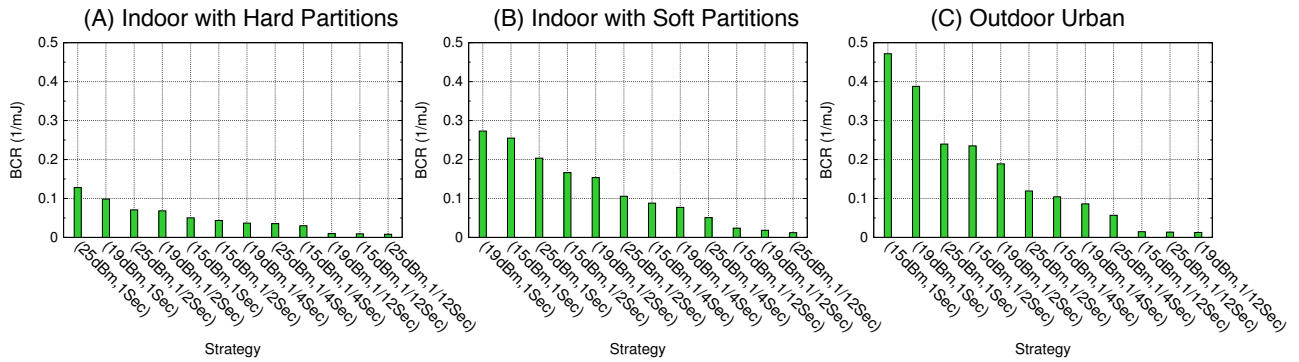


Fig. 3.13: Strategies ranked in the descending order with respect to their corresponding BCR in different environments

improvement becomes less significant as the environment becomes less obstructed. Accordingly, increasing  $pow_{tx}$  can lead to a decrease in BCR in less obstructed environments.

- *Decreasing  $\Delta_{period}$  for a given  $pow_{tx}$ .* For a fixed value of  $pow_{tx}$ , decreasing  $\Delta_{period}$  decreases the value of BCR regardless of the environment (see Fig. 3.12). In fact, we know that decreasing  $\Delta_{period}$  increases the energy consumption considerably. We also know that decreasing  $\Delta_{period}$  can have different impacts on the detection probability depending on the environment. In particular, in indoor environments, decreasing  $\Delta_{period}$  down to some value ( such as 1/4 second in *indoor with hard partitions* environment ) can improve the detection probability. However, in all these cases the loss in energy consumption is so significant that it mitigates the potential gain in detection probability.

### 3.6.5 The Tradeoff Strategy

Fig. 3.13 on page 90, shows the strategies ranked in descending order with respect to their corresponding BCR in different environments. As already discussed in Section 3.6.4, decreasing  $\Delta_{period}$  decreases BCR in all cases. That is why the tradeoff strategy has  $\Delta_{period} = 1$  second in all environments. Moreover, increasing  $pow_{tx}$  improves BCR only when the environment is obstructed and as the environment becomes less obstructed it can even decrease BCR. That is why the tradeoff strategy in *indoor with hard partitions*, *indoor with soft partitions* and *outdoor urban* environments has respectively  $pow_{tx}$  of 25 dBm, 19 dBm and 15 dBm.

### 3.6.6 How Effective and Efficient is the Tradeoff Strategy?

A summary of the results of BCR evaluation can be found in Tables 3.4 and 3.5 on page 92. As shown in Table 3.4, in all environments, the effectiveness ratio of the tradeoff strategy is high and close to 1. More precisely, the effectiveness ratio of the tradeoff strategy is equal to 0.90, 0.76 and 0.88 in *indoor with hard partitions*, *indoor with soft partitions* and *outdoor urban areas*, respectively. Accordingly, the tradeoff strategy has a relatively good effectiveness rank i.e., 3rd, 6th and 6th in *indoor with hard partitions*, *indoor with soft partitions* and *outdoor urban areas*, respectively.

Table 3.4: Comparison of the most effective strategy, the most efficient strategy and the tradeoff strategy in different environments

Environment	Strategy	Detection Probability	Energy Consumption (mJ)	BCR (1/mJ)	Effectiveness Ratio	Efficiency Ratio	Effectiveness Rank	Efficiency Rank
Indoor-hard	<b>The Most Effective Strategy</b> (25 dBm, 1/4 Sec)	0.5722	16.15	0.035	1	0.11	1	9
	<b>The Most Efficient Strategy</b> (15 dBm, 1 Sec)	0.0910	1.809	0.050	0.16	1	12	1
	<b>The Tradeoff Strategy</b> (25 dBm, 1 Sec)	0.5157	4.037	0.127	0.90	0.44	3	4
Indoor-soft	<b>The Most Effective Strategy</b> (25 dBm, 1/2 Sec)	0.8536	8.07	0.105	1	0.22	1	7
	<b>The Most Efficient Strategy</b> (15 dBm, 1 Sec)	0.4612	1.809	0.254	0.54	1	12	1
	<b>The Tradeoff Strategy</b> (19 dBm, 1 Sec)	0.6539	2.39	0.273	0.76	0.75	6	2
Outdoor-urban	<b>The Most Effective Strategy</b> (25 dBm, 1 Sec)	0.9679	4.04	0.239	1	0.44	1	4
	<b>The Most Efficient Strategy</b> (15 dBm, 1 Sec)	0.8532	1.809	0.471	0.88	1	6	1
	<b>The Tradeoff Strategy</b> (15 dBm, 1 Sec)	0.8532	1.809	0.471	0.88	1	6	1

Table 3.5: Impact of changing  $pow_{tx}$  or  $\Delta_{period}$  on effectiveness, efficiency and BCR in different environments

Environment	Increasing $pow_{tx}$			Decreasing $\Delta_{period}$		
	Impact on effectiveness	Impact on efficiency	Impact on BCR	Impact on effectiveness	Impact on efficiency	Impact on BCR
Indoor-hard	Improves	Deteriorates	Improves only under $\Delta_{period} = 1$ second.	Generally improves if $\Delta_{period}$ is decreased down to 1/4 second.	Deteriorates	Deteriorates
Indoor-soft	Improves	Deteriorates	Improves only under $\Delta_{period} = 1$ second and when $pow_{tx}$ is increased up to 19 dBm.	Generally improves if $\Delta_{period}$ is decreased down to 1/2 second.	Deteriorates	Deteriorates
Outdoor-urban	Improves	Deteriorates	Deteriorates	Deteriorates	Deteriorates	Deteriorates

Moreover, the efficiency ratio of the tradeoff strategy is equal to 0.44, 0.75 and 1 in *indoor with hard partitions*, *indoor with soft partitions* and *outdoor urban areas*, respectively (see Table 3.4). In fact,  $\Delta_{period}$  of the tradeoff strategy is equal to 1 second in all environments and we know that a high value of  $\Delta_{period}$  results in low energy consumption. At the same time,  $pow_{tx}$  of the tradeoff strategy decreases as the environment becomes less obstructed which causes the decrease of its energy consumption in less obstructed environment and increase of its efficiency ratio. Thereby, the tradeoff strategy has a relatively good efficiency rank i.e., 4th, 2nd and 1st in *indoor with hard partitions*, *indoor with soft partitions* and *outdoor urban areas*, respectively. Note that in the *outdoor urban* environment, the tradeoff strategy is the same as the most efficient strategy, and therefore it has the lowest energy consumption compared to other strategies.

### 3.7 Related Work

The existing studies on transmit power control in ad hoc networks [27], for the most part, consider the unicast communications and therefore are not relevant to our work. Also, most of the papers which study the problem of reliable broadcast in MANETs, consider one-shot broadcast and not periodic broadcast [21]. For these reasons, in this section we only discuss the works performed in the two fields which we believe are the closests to our work, i.e., *enhancements of the hello protocol*; and *beacon broadcast for VANET safety applications*.

#### 3.7.1 Enhancements of the hello Protocol

Neighbor detection in ad hoc networks is usually studied as a building block for applications such as routing, leader election, group management and localization. The neighbor detection algorithm introduced in this paper is inspired by the *basic hello protocol* first described in *Open Shortest Path First (OSPF)* routing protocol [35]. It works as follows. Nodes periodically send *hello* messages to announce their presence to close nodes, and maintain a neighborhood table. The sending frequency is denoted by  $f_{hello}$ . Thus, in the case of our neighbor detection algorithm  $f_{hello} = 1/\Delta_{period}$ .

In the literature, there exist several works that attempt to improve the basic hello protocol performance especially with regards to energy consumption. These works can be classified into two categories described below.

The first category consists of the algorithms that aim at minimizing the energy consumption by keeping the nodes in the sleep radio mode most of the time and thus, reaching neighbor detection with as few transmission and/or reception attempts as possible [34; 42; 51; 17; 48; 26; 6]. The majority of the enhancements of the hello protocol belongs to this category. However, as described in Section 3.5.1, we do not consider the sleep radio mode in this paper. Therefore, we do not discuss the works in this category here.

The second category (which is more relevant to our work) consists of the algorithms that aim at minimizing the energy consumption of neighbor detection by (regularly) adapting  $f_{hello}$  and/or the transmission power to the network changes [22; 24; 23]. For instance, in the protocol proposed in [22], the value of  $f_{hello}$  is adapted based on link connectivity. More precisely, the authors define two metrics: Time to Link Failure (TLF) and Time Without link Changes (TWC). Thus, each node regularly evaluates these metrics and compares them with some given thresholds. Then, based on the comparisons, the node can switch  $f_{hello}$  from a high value ( $f_{high}$ ) to a low value ( $f_{low}$ ) and vice-versa. The drawback of this approach is that the thresholds may need to evolve over time and finding the correct thresholds is not obvious.

The protocols in [24; 23] adapt  $f_{hello}$  to the variations of node speed. They are based on the existence of an optimal *hello* frequency denoted by  $f_{opt}$ . The equation for  $f_{opt}$  was first introduced in [47] where it is calculated based on the relative speed of nodes. In [24], the authors proposed a Turnover based Adaptive hello Protocol (TAP). According to TAP, each node evaluates the changes of its neighborhood table periodically (i.e., before sending each *hello*) and calculates the turnover. The turnover (also denoted by  $r$ ) is, in fact, the ratio between the number of new neighbors (i.e., nodes detected during the last period) and the current total number of neighbors. Once the turnover  $r$  is calculated, it is compared to  $r_{opt}$  where  $r_{opt}$  is the turnover corresponding to  $f_{opt}$ . If  $r < r_{opt}$ , this means that  $f_{hello}$  is too high and there are not enough changes in the table. Therefore,  $f_{hello}$  should be decreased. On the contrary, if  $r > r_{opt}$ , this means that  $f_{hello}$  is too low and that there are too many changes. Therefore,  $f_{hello}$  should be increased. In this way, TAP keeps the value of  $f_{hello}$  always close to  $f_{opt}$ . In [23] a Turnover based Frequency and transmis-



sion Power Adaptation algorithm (TFPA) is presented. TFPA applies a similar (but slightly improved) approach as TAP to dynamically adapt  $f_{hello}$ . Moreover, TFPA calculates an optimal value for transmission range based on the energy consumption model presented in [20]. Then, it dynamically adapts the transmission power so that the resulting transmission range remains close to the optimal value. Compared to TAP, TFPA has a lower energy consumption since it dynamically adapts both  $f_{hello}$  and the transmission power. However, both protocols have assumptions which are less realistic compared to the assumptions in our work. More precisely, they consider the unit disk graph and a deterministic radio propagation model whereas we use a probabilistic radio propagation model. Moreover, the energy consumption model assumed by TFPA, corresponds to sensor nodes and does not realistically reflect the energy consumption of WNICs used in today’s mobile devices whereas we model the energy consumption based on specifications of current smartphones. Finally, both protocols use  $f_{opt}$  which is basically determined by the relative speed of nodes. As a result, they essentially adapt  $f_{hello}$  according to the speed changes and neglect the other factors such as the environment attenuation or congestions. In our work, on the other hand, we take into account the environment attenuations as well as interferences and collisions and neglect the relative speed of the nodes. In fact this choice is justified by the fact that in our case the devices are usually used by pedestrians. Hence, they have a slow movement especially during the neighborhood time of 4 seconds.

### 3.7.2 Beacon Broadcast for VANET Safety Applications

In vehicular ad hoc networks (VANETs), safety applications aim at minimizing accident levels. One type of safety messages are beacons. A beacon usually contains the vehicle’s position, speed, direction, etc and is periodically broadcast in a single-hop manner. By using beacons, safety applications gain knowledge of the surroundings and can prevent dangerous situations for the drivers. The safety applications usually require the beacon delivery up to a certain distance and have some guarantees for message reliability and latency. Thus, many papers study the impact of parameters such as transmission power, packet generation rate or packet size on fulfillment of applications’ guarantees [54; 45; 46; 32].

For instance, in [45] authors present a simulation-based study in ns-2.28 to analyze the impact of transmission power and packet generation rate on the reception of beacon messages. Authors consider 1800 nodes uniformly distributed in a circular map. All nodes broadcast messages with common transmission power and packet generation rate. The broadcast reception is only studied for messages of senders located at 40 m from a receiver which is located at the center of the map. Regarding the impact of transmission power on broadcast reception, authors state that the transmission power should be strong enough to resist the interferences but not so strong as to increase the load on the medium. Regarding the impact of packet generation rate on broadcast reception, they state that increasing packet generation rate can increase the number of received packets significantly, as long as the channel busy time (or the time ratio a node determines the channel as busy) has not reached its maximum. There are several differences between this study and ours: firstly, this study is performed for VANETs. Thus, the 802.11 physical layer parameters are set to the specific values of the *802.11p* standard whereas we use the values of the 802.11a standard. Secondly, this study uses the *Nakagami* radio propagation model known to model signal attenuation in VANETs, whereas we use LNS to model obstructed urban areas. Thirdly, this study uses different metrics (such as channel busy time) than our metrics to interpret the results. Finally, this study does not consider the impact of transmission power or packet generation rate on the energy consumption.

In [46], authors show that periodic beacon transmission can result in the channel saturation which in turn, causes a high number of packet collisions and low reception rates. They also show that simply increasing the packet generation rate or the transmission power can exacerbate the channel conditions. Therefore, they propose an algorithm called Distributed Fair Power Adjustment for Vehicular environments (D-FPAV). The algorithm limits the beaconing load on the channel below a predefined threshold while ensuring a high probability of beacon delivery at close distances from the sender. Similarly to the work in [45], this work also considers the *802.11p* physical and MAC layers and the *Nakagami* radio propagation model and thus has a different system model compared to our work. However, the basic idea behind the D-FPAV algorithm is to fix the packet generation rate at the minimum required by safety applications, and to adjust the transmission power of beacons in case of congestion. This means that the way that the D-FPAV algorithm adapts the transmission power and the packet generation rate somehow resembles the way that

our tradeoff strategy is adapted in different environments. In fact, for the tradeoff strategy we keep  $\Delta_{period}$  at its highest value and we decrease the transmission power in less obstructed environments i.e., where the collisions and interferences are high.

### 3.8 Conclusion

To the best of our knowledge, this is the first paper that studies the impact of transmission power and broadcast interval on effectiveness and efficiency of neighbor detection for MANETs in different urban environments. Our results can be used as a basis to design adaptive neighbor detection algorithms for urban environments. Such algorithms can adapt the transmission power and broadcast interval based on environment and application guarantees on effectiveness and efficiency. To deploy such algorithms on a smartphone, one can use lightweight sensing services such as the one introduced in [56] which can detect the indoor/outdoor environment in a fast, accurate, and efficient manner.

Relying on a realistic simulation-based study, we showed that the most effective strategy is not the same as the most efficient strategy in any environment. In fact, in all environments, there is a conflict between effectiveness and efficiency such that the most effective strategy is usually not very efficient and the most efficient strategy is not always very effective. However, we showed that the conflict becomes less severe as the environment becomes less obstructed. When discussing our results, we also described how a change in transmission power and broadcast interval can influence the effectiveness and efficiency. We then proposed an approach to make a tradeoff between effectiveness and efficiency. Accordingly, we identified the tradeoff strategy in each environment and we showed that it has a relatively good effectiveness and efficiency compared to other strategies.

The conclusions drawn in this paper could still be more realistic if our evaluations could be performed on a real prototype. In fact, we are currently developing, *ManetLab*, a modular and configurable software framework for creating and running testbeds to evaluate MANET-specific protocols [50]. Thus, as a potential future work, we consider to deploy the neighbor detection algorithm on *ManetLab* and compare the results of our evaluations with the ones performed using the *ManetLab*.

There are also some issues which remain open and we might consider as future work. For instance, the possibility to detect nodes using an underlying multi-hop network should be investigated. Moreover, in this paper we did not consider the sleep radio mode for 802.11 communication. Since for energy consumption we only took into account the active energy, it seems that our results could still be valid for the case when the sleep radio mode is enabled. Thus, this issue can also be investigated as future work.

## References

- [1] IEEE 802.11: Wireless LAN MAC and Physical Layer Specifications, 1999.
- [2] IEEE std 802.11a/D7.0-1999, part11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications: high speed physical layer in the 5 GHz band, 1999.
- [3] Official IEEE 802.11 working group project timelines, Available: [http://grouper.ieee.org/groups/802/11/Reports/802.11\\_Timelines.htm](http://grouper.ieee.org/groups/802/11/Reports/802.11_Timelines.htm), 2014.
- [4] Apple iPhone 4S Teardown. Available: <http://www.techinsights.com/teardowns/apple-iphone-4s-teardown/>
- [5] F. Bai and A. Helmy, A survey of mobility modeling and analysis in wireless ad hoc networks, Book Chapter In *Wireless ad hoc and sensor networks*, Springer, 2006.
- [6] M. Bakht, M. Trower, and R. Kravets. Searchlight: helping mobile devices find their neighbors. In *ACM SIGOPS Oper. Syst.*, vol. 45, no. 3, pp. 71–76, 2012.
- [7] S. Basagni, M. Conti, S. Giordano and I. Stojmenovic. Mobile ad hoc networking: the cutting edge directions, Wiley-IEEE Press, 2013.
- [8] Best iPhone bluetooth games. <http://iphone.mob.org/genre/multipleer/>
- [9] N. E., Boardman, Cost-benefit analysis: concepts and practice (3rd ed.). Upper Saddle River, NJ, Prentice Hall, 2006.
- [10] B. Bostanipour, B. Garbinato and A. Holzer, Spotcast – A communication abstraction for proximity-based mobile applications, In *Proc. IEEE NCA'12*, pp. 121–129, 2012.
- [11] B. Bostanipour and B. Garbinato, Improving neighbor detection for proximity-based mobile applications, In *Proc. IEEE NCA'13*, pp. 177–182, 2013.

- [12] B. Bostanipour and B. Garbinato, Using virtual mobile nodes for neighbor detection in proximity-based mobile applications, In *Proc. IEEE NCA'14*, pp. 9–16, 2014.
- [13] Broadcom. <http://www.broadcom.com/>
- [14] M. M. Carvalho, C. B. Margi, K. Obraczka, and J. J. Garcia-Luna-Aceves, Modeling energy consumption in single-hop IEEE 802.11 ad hoc networks, In *Proc. IEEE ICCCN'04*, pp. 367–372, 2004.
- [15] Q. Chen, F. Schmidt-Eisenlohr, D. Jiang, M. Torrent-Moreno, L. Delgrossi, and H. Hartenstein, Overhaul of IEEE 802.11 modeling and simulation in NS-2, In *Proc. ACM MSWiM'07*, pp. 159–168, 2007.
- [16] R. Coté and G. E. Harrington, Life safety code handbook (NFPA 101), 2009.
- [17] P. Dutta, D. Culler, Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proc. ACM SenSys'08*, 2008.
- [18] J.-P. Ebert, B. Burns and A. Wolisz, A trace-based approach for determining the energy consumption of a WLAN network interface, In *Proc. of European Wireless*, pp. 230–236, 2002.
- [19] L. M. Feeney and M. Nilsson, Investigating the energy consumption of a wireless network interface in an ad hoc networking environment, In *Proc. IEEE INFOCOM'01*, pp. 1548–1557, 2001.
- [20] E. Fleury and D. Simplot-Ryl. Réseaux de capteurs. Hermes Science - Lavoisier, 2009.
- [21] B. Garbinato, A. Holzer, F. Vessaz, Context-aware broadcasting approaches in mobile ad hoc networks, In *Comput. Netw.*, vol. 54, no. 7, pp. 1210–1228, 2010.
- [22] C. Gomez, A. Cuevas, and J. Paradells, AHR: a two-state adaptive mechanism for link connectivity maintenance in AODV, In *Proc. ACM REALMAN'06*, 2006.
- [23] D. He, N. Mitton, and D. Simplot-Ryl, An energy efficient adaptive HELLO algorithm for mobile ad hoc networks, In *Proc. ACM MSWiM'13*, pp. 65–72, 2013.
- [24] F. Ingelrest, N. Mitton, and D. Simplot-Ryl, A turnover based adaptive hello protocol for mobile ad hoc and sensor networks, In *Proc. MASCOTS'07*, pp. 9–14, 2007.
- [25] J.-R. Jiang, Y.-C. Tseng, C.-S. Hsu, and T.-H. Lai, Quorum-based asynchronous power-saving protocols for IEEE 802.11 ad hoc networks, In *ACM Mob. Netw. Appl.*, vol. 10, no. 1–2, pp. 169–181, 2005.

- [26] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar, U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol, In *IPSN'10*, pp. 350–361, 2010.
- [27] V. Kawadia and P. Kumar, Principles and protocols for power control in wireless ad hoc networks, In *IEEE JSAC*, vol. 23, no. 1, pp. 76–88, 2005.
- [28] V. Kelly, New IEEE 802.11ac specification driven by evolving market need for higher, multi-user throughput in wireless LANs, IEEE, 2014.
- [29] J. Lee, W. Kim, S-J Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi, An experimental study on the capture effect in 802.11a networks. In *Proc. ACM WinTECH'07*, pp. 19–26, 2007.
- [30] J. Li, J. Xiao, J. W.-K. Hong and R. Boutaba, Application-centric Wi-Fi energy management on smart phone, In *Proc. IEEE APNOMS'12*, pp. 1–8, 2012.
- [31] LoKast. <http://www.lokast.com/>
- [32] X. Ma, J. Zhang, and T. Wu, Reliability analysis of one-hop safety-critical broadcast services in VANETs, *IEEE Trans. Vehicular Tech.*, vol. 60, no.8, pp. 3933–3946, 2011.
- [33] J. Manweiler and R. R. Choudhury, Avoiding the rush hours: WiFi energy management via traffic isolation, *IEEE Trans. on Mobile Computing*, vol. 11, no. 5, pp. 739-752, 2012.
- [34] M. J. McGlynn and S. A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proc. ACM MobiHoc'01*, pp. 137–145, 2001.
- [35] J. Moy, OSPF – Open Shortest Path First, RFC 1583, 1994.
- [36] T. Muetze, P. Stuedi, F. Kuhn and G. Alonso, Understanding radio irregularity in wireless networks, In *Proc. IEEE SECON'08*, pp. 82–90, 2008.
- [37] ns-2.35 (released on Nov. 4, 2011). <http://www.isi.edu/nsnam/ns/>
- [38] D. Qiao, S. Choi, A. Jain, and K. G. Shin, Miser: an optimal low-energy transmission strategy for IEEE 802.11a/h, In *Proc. ACM MobiCom'03*, pp. 161–175, 2003.
- [39] K. Ramachandran, R. Kokku, H. Zhang and M. Gruteser, Symphony: synchronous two-phase rate and power control in 802.11 WLANs, *IEEE/ACM Trans. on Networking*, vol.18, no.4, pp. 1289–1302, 2010.
- [40] T. S. Rappaport, Wireless communications: principles and practice, 2nd Edition, Prentice Hall, 2002.

- [41] M. Raya and J. P. Hubaux, The security of vehicular ad hoc networks, In *Proc. ACM SASN'05*, pp. 11–21, 2005.
- [42] G. Sawchuk, G. Alonso, E. Kranakis, and P. Widmayer. Randomized protocols for node discovery in ad hoc multichannel networks. In *Proc. Ad Hoc Now*, 2003.
- [43] J. Schiller, Mobile communications, 2nd Edition, Addison-Wesley, England, 2003.
- [44] F. Schmidt-Eisenlohr. Interference in vehicle-to-vehicle communication networks –analysis, modeling, simulation and assessment. PhD thesis, Karlsruhe Institute of Technology (KIT), 2010.
- [45] M. Torrent-Moreno, S. Corroy, F. Schmidt-Eisenlohr, and H. Hartenstein, IEEE 802.11-based one-hop broadcast communications: understanding transmission success and failure under different radio propagation environments, In *Proc. ACM MSWiM'06*, pp. 68–77, 2006.
- [46] M. Torrent-Moreno, J. Mittag, P. Santi, and H. Hartenstein, Vehicle-to-Vehicle communication: fair transmit power control for safety-critical information, *IEEE Trans. Vehicular Tech.*, vol. 58, pp. 3684–3703, 2009.
- [47] A. Troël. Prise en compte de la mobilité dans les interactions de proximité entre terminaux à profils hétérogènes. PhD thesis, Université de Rennes, 2004.
- [48] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh. Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks. In *Proc. IEEE INFOCOM'02*, 2002.
- [49] W. Vandenberghe, I. Moerman, and P. Demeester, On the feasibility of utilizing smartphones for vehicular ad hoc networking, In *Proc. IEEE ITST'11*, pp. 246–251, 2011.
- [50] François Vessaz, Benoît Garbinato, Arielle Moro and Adrian Holzer, Developing, deploying and evaluating protocols with ManetLab. In *Proc. NETYS'13*, pp. 89–104, 2013.
- [51] G. Wattenhofer, G. Alonso, E. Kranakis, and P. Widmayer. Randomized protocols for node discovery in ad hoc, single broadcast channel networks. In *Proc. IPDPS'03*, 2003.
- [52] Waze. <https://www.waze.com/en/>
- [53] WhosHere. <http://whoshere.net/>
- [54] Q. Xu, T. Mak, J. Ko, and R. Sengupta, Vehicle-to-vehicle safety messaging in DSRC, In *Proc. ACM VANET'04*, pp. 19–28, 2004.
- [55] R. Zheng, J. C. Hou and L. Sha, Asynchronous wakeup for ad hoc networks, In *Proc. ACM MobiHoc'03*, pp. 35–45, 2003.

- [56] P. Zhou, Y. Zheng, Z. Li, M. Li, and G. Shen, IODetector: a generic service for indoor outdoor detection. In *Proc. ACM SenSys'12*. pp 113–126, 2012.



## Chapter 4

# Using Virtual Mobile Nodes for Neighbor Detection in Proximity-Based Mobile Applications

**Abstract** We introduce a time-limited neighbor detector service for mobile ad hoc networks, which enables a mobile device to detect other nearby devices in the past, present and up to some bounded time interval in the future. Our motivation lies in the emergence of a new trend of mobile applications known as proximity-based mobile applications, which enable a user to communicate with other users within some defined range and for a certain amount of time. Neighbor discovery is a fundamental requirement for these applications and is not restricted to the current neighbors but can include past or future neighbors. To implement the time-limited neighbor detector service, we apply an approach based on virtual mobile nodes. A virtual mobile node is an abstraction that is akin to a mobile node that travels in the network in a predefined trajectory. In practice it can be implemented by a set of mobile nodes based on a replicated state machine approach. In this paper, we assume that each node can accurately predict its own locations up to some bounded time interval in the future. Thus, we present a time-limited neighbor detector algorithm that uses a virtual mobile node that continuously travels in the network, collects the predicted locations of all nodes, performs the neighborhood matching between nodes and sends the list of neighbors to each node. We show that our algorithm correctly implements the time-limited neighbor detector service under a set of conditions.

### **Publication:**

B. Bostanipour and B. Garbinato, Using virtual mobile nodes for neighbor detection in proximity-based mobile applications, *In Proceedings of The 13th IEEE International Symposium on Network Computing and Applications (IEEE NCA '14)*, pp. 9–16, Cambridge, Massachusetts, USA, IEEE, 2014.

## 4.1 Introduction

With the ubiquitous use of mobile devices and particularly smartphones, we face the emergence of a new type of distributed applications known as *Proximity-Based Mobile (PBM)* applications [5; 6]. These applications enable a user to interact with others in a defined range and for a given time duration e.g., for social networking ([26; 21; 1; 20]), gaming ([4]) and driving ([25]).

Discovering who is nearby is at the core of the PBM applications. It is the preliminary step for the further interactions between users. It also provides users with the opportunity to extend their social network from the people that they know to the people that they might not know but who are in their proximity. For instance, in a simple usage scenario of social networking applications such as WhosHere [26] or LoKast [21], a user first discovers other users in her proximity and then decides to view their profiles, start a chat with a user or a group of users or add them as friends. The discoverability, however, may not always be limited to the current neighbors. For instance, with the social networking applications such as iGroups [1] or LocoPing [20], a user can discover other users who were in her vicinity during a past event (e.g., concert, tradeshow, wedding) or simply during a past time interval (e.g., the past 24 hours). One can also think of applications that provide the user with the list of people who will be in her proximity up to some time interval in future and thus create the potential for new types of social interactions.

In this paper, we introduce a *time-limited neighbor detector* service that enables a user to discover the set of its neighbors in the past, present and up to some bounded time interval in the future in a mobile ad hoc network (MANET). We also present an algorithm that implements the time-limited neighbor detector using a virtual mobile node. A virtual mobile node is an abstraction that is already introduced in the literature and used for tasks such as routing or collecting data in MANETs [10; 11]. It is akin to a mobile node that travels in the network in a predefined trajectory known in advance to all nodes. In practice a virtual mobile node is emulated by a set of nodes in the network based on a replicated state machine approach. In this paper, we assume that each node can accurately predict its own locations up to some bounded time interval in the future. Thus, in the algorithm we use the virtual mobile node to continuously travel in the network, collect the the predicted locations of all nodes, perform the neighborhood matching between nodes and then send back to each node the list of its neighbors at current and future times. Each node also

stores its neighbor set at any time so that later it will be able to be queried about its past neighbors.

In order to show that our algorithm correctly implements the time-limited neighbor detector service, we present a proof of correctness. In particular, we define the conditions under which the service can be correctly implemented by the algorithm.

To the best of our knowledge, this is the first paper that introduces a neighbor detector service that can detect future neighbors (although up to some time interval) in a MANET. It is also the first paper that uses an approach based on virtual mobile nodes for neighbor discovery.

The remainder of the paper is as follows. In Section 4.2, we describe our system model and present some definitions. In Section 4.3, we introduce the neighbor detector abstraction in two variants: the *perfect* variant which presents the ideal case of neighbor detection and is rather impractical and the *time-limited* variant. In Section 4.4, we present the implementation of the time-limited variant of the neighbor detector service. In order to do so, we first describe what is a virtual mobile node and how it can be used for the implementation of the time-limited neighbor detector. We then introduce an algorithm that implements the time-limited neighbor detector and we present a proof of correctness for the algorithm. In Section 4.5, we discuss the related work. Finally, in Section 4.6, we discuss open problems and future work.

## 4.2 System Model and Definitions

We consider a mobile ad-hoc network (MANET) consisting of processes that move in a bounded region  $R$  of a two-dimensional plane. We use the terms *process* and *node* interchangeably. Each process is assigned a unique identifier. Processes can move on any continuous path, however there exists a known upper bound on their movement speed. A process is prone to *crash-reboot* failures: it can fail and recover at any time, and when the process recovers, it returns to its initial state. Moreover, a process may join or leave the system at any time (where a leave is treated as a failure). A process is *correct* if it never fails. Since we do not consider *Byzantine* behaviors, the information security and privacy issues are beyond the scope of this paper.

We assume the existence of a discrete global clock, i.e., the range  $T$  of the clock's ticks is the set of non-negative integers. We also assume the existence of a known

bound on the relative processing speed. Each process in the system has access to a *global positioning service*, a *timely scoped broadcast service* and a *mobility predictor service*. In the following, we first introduce some definitions that are used throughout the paper. We then present each of the above mentioned services.

### 4.2.1 Definitions

Let  $p_i$  be a process in the network, we introduce the definitions given hereafter in order to capture proximity-based semantics.

- location denotes a geometric point in the two dimensional plane where region  $R$  is situated and can be expressed as tuple  $(x, y)$ .
- $loc_i(t)$  denotes the location occupied by process  $p_i$  at time  $t \in T$ .
- $Z_i(r, t)$  denotes all the locations inside or on the circle centered at  $loc_i(t)$  with given radius  $r$ .
- $r_d$  is called *neighbor detection radius*. It is a constant known by all processes in the network. Thus,  $Z_i(r_d, t)$  presents the *neighborhood region* of  $p_i$  at time  $t$ .

### 4.2.2 Timely Scoped Broadcast Service

This communication service allows a process to send messages to all processes located within a given radius around it. Formally, the timely scoped broadcast service exposes the following primitives:

- $BROADCAST(m, r)$ : broadcasts a message  $m$  in  $Z_i(r, t_b)$ , where  $p_i$  is the sender and  $t_b$  is the time when the broadcast is invoked.
- $RECEIVE(m, p_i)$ : callback delivering a message  $m$  broadcast by process  $p_i$ .

The service satisfies the following properties.

**Timely Delivery.** If a correct process  $p_i$  broadcasts a message  $m$ , there exists a bounded time duration  $\Delta_{bcast}$  such that every correct process  $p_j$  delivers  $m$  in interval  $[t_b; t_b + \Delta_{bcast}]$ , if  $loc_j(t) \in Z_i(r, t)$  for all  $t \in [t_b; t_b + \Delta_{bcast}]$ .

**No Duplication.** No message is delivered more than once.

**No Creation.** If some process  $p_j$  delivers a message  $m$  with sender  $p_i$ , then  $m$  was previously broadcast by process  $p_i$ .

For a detailed discussion regarding the implementability of this service in both the single-hop and the multi-hop cases see [5].

### 4.2.3 Global Positioning Service

This service allows each mobile process  $p_i$  to know its current location and the current time via the following functions:

- **GETCURRENTTIME**: returns the current global time. Formally, this implies that each process  $p_i$  has access to the global clock modeled in Section 4.2.
- **GETCURRENTLOCATION**: returns the location occupied by  $p_i$  at the current global time.

In this paper, we do not provide any formal properties for this service. However, we assume that the outputs of its functions are exact. In practice, such a service would typically be implemented using NASA’s GPS or ESA’s Galileo space-based satellite navigation technologies.

### 4.2.4 Mobility Predictor Service

This service allows each mobile process  $p_i$  to predict its future locations up to some bounded time  $\Delta_{predict}$  via the following function:

- **PREDICTLOCATIONS**: returns a hash map containing the predicted locations for  $p_i$  at each time  $t$  in the interval  $[t_c; t_c + \Delta_{predict}]$  where  $t_c$  is the time when **PREDICTLOCATIONS** is invoked.

The service satisfies the following property.

**Strong Accuracy.** Let  $t \in [t_c; t_c + \Delta_{predict}]$  and  $l$  be a location, if  $p_i$  is predicted to be at  $l$  at time  $t$ , then  $loc_i(t) = l$ .

In order for the service to predict the locations of the process in the future, we assume that the mobility model applied by the processes is such that the future locations of a process can be predicted up to a certain  $\Delta_{predict}$ . For instance, if a process moves according to a mobility model with temporal dependency (such

as Gauss-Markov Mobility Model or Smooth Random Mobility Model), its future locations can be predicted using its past locations [2]. In practice, the future location prediction can be achieved for example by taking into account the current trajectory (direction, speed,...) of the mobile device, possibly coupled with maps information.

### 4.3 The Neighbor Detector Abstraction

In this section, we introduce the *neighbor detector* abstraction in two variants. Formally, the neighbor detector service exposes the following primitive:

- **PRESENT**( $t$ ): returns  $N_i(t)$  i.e., the set of processes detected as neighbors of  $p_i$  at time  $t$ , where  $p_i$  is the process that invokes **PRESENT**.

#### 4.3.1 Neighbor Detector Variants

##### 4.3.1.1 Perfect Neighbor Detector

By querying this variant of neighbor detector service, a mobile process is able to know the set of its neighbors at any time in the past, present or the future.

**Perfect Completeness.** Let  $p_i$  and  $p_j$  be two correct processes, if  $loc_j(t) \in Z_i(r_d, t)$ , then  $p_j \in N_i(t)$ .

**Perfect Accuracy.** Let  $p_i$  and  $p_j$  be two correct processes, if  $p_j \in N_i(t)$ , then  $loc_j(t) \in Z_i(r_d, t)$ .

Roughly speaking, the *perfect completeness* property requires a neighbor detector to detect any node that is in the neighborhood region at any time in the past, present or future. At the same time, the *perfect accuracy* property guarantees that no false detection occurs. Since in practice implementing the *perfect completeness* property requires an infinite knowledge of nodes' locations in the future, we consider a more practical variant of the neighbor detector abstraction called *the time-limited neighbor detector*. We introduce this variant hereafter.

#### 4.3.1.2 Time-limited Neighbor Detector

Compared to the *perfect neighbor detector*, this variant has a different *completeness* property. However, its *accuracy* property is the same. We define its properties, below.

**Time-limited Completeness.** Let  $p_i$  and  $p_j$  be two correct processes and  $\Delta_{future}$  be a bounded time interval such that  $\Delta_{future} > 0$ , if  $loc_j(t) \in Z_i(r_d, t)$  and  $t \leq t_c + \Delta_{future}$ , then  $p_j \in N_i(t)$ , where  $t_c$  is the time when PRESENT is invoked at  $p_i$ .

**Perfect Accuracy.** Let  $p_i$  and  $p_j$  be two correct processes, if  $p_j \in N_i(t)$ , then  $loc_j(t) \in Z_i(r_d, t)$ .

Similarly to the *perfect completeness* property, the *time-limited completeness* property requires a neighbor detector to detect any node that is in the neighborhood region at any time in the past or present. However, its ability to detect future neighbors is limited by a bounded time duration  $\Delta_{future}$ . More precisely, it only detects a node that is in the neighborhood region at any time from the time when PRESENT is invoked up to  $\Delta_{future}$ . The *perfect accuracy* property also guarantees no false detection.<sup>1</sup>

### 4.4 Implementing The Time-Limited Neighbor Detector

To implement the time-limited neighbor detector, our intuition is as follows: since each node knows its own locations up to  $\Delta_{predict}$  in the future, we can think of a moving entity that travels through the network, collects the predicted locations of all nodes, performs the neighborhood matching between nodes and then sends back to each node the list of its neighbors at current and future times. For this reason, we rely on the concept of *Virtual Mobile Node* (VMN) introduced in [10].

In what follows we first describe what is a VMN and we add a VMN to the system model. We then introduce an algorithm that implements the time-limited neighbor detector in the new system model. Finally, we discuss the correctness of the algorithm.

---

<sup>1</sup> For simplicity's sake, we do not assume a time bound on the availability of the past neighborhood information at this point.

### 4.4.1 Virtual Mobile Node (VMN)

A VMN is an abstraction that is akin to a mobile node that travels in the network in a predefined trajectory. It is first introduced in [10]. One of the main motivations behind the design of a VMN abstraction is that if the motion of a mobile node could be predefined and known to all nodes in the network, the task of designing various algorithms for mobile ad hoc networks would be significantly simplified. Therefore, a VMN is designed such that it can execute any distributed algorithm that a node can execute, however, its movement can be predefined and be known in advance to all nodes in the network.

In [10] an algorithm called *Mobile Point Emulator (MPE)* is introduced which implements the VMN abstraction in a system model equivalent to the system model defined in this paper. Its approach to implement the VMN is based on a replicated state machine technique similar to the one originally presented in [19]. The algorithm defines a *mobile point* to be a circular region of a radius  $r_{mp}$ , that moves according to the predefined path of the VMN, i.e., at time  $t$  the center of the mobile point coincides with the preplanned location of the VMN at time  $t$ . The MPE replicates the state of the VMN at every node within the mobile point's region, modifying the set of replicas as the nodes move in and out of the mobile point's region. MPE uses a total-order broadcast service to ensure that the replicas are updated consistently. The total order broadcast service is built using a synchronous local broadcast service (equivalent to our timely scoped broadcast) and the synchronized clocks (obtained by using a service equivalent to our global positioning service).

Similarly to a node, the VMN can communicate with other nodes using the timely scoped broadcast service (or its equivalents). Moreover, the VMN is prone to *crash-reboot* failures. It can crash if and only if its trajectory takes it into a region unpopulated by any nodes (i.e., where there are no nodes to act as replicas), however, it recovers to its initial state as soon as it reenters a dense area. A VMN is *correct* if it never fails.

### 4.4.2 Adding a VMN to the System Model

In this section, we add a VMN to the system model defined in Section 4.2. The movement trajectory of this VMN is such that it can be used by our algorithm for the



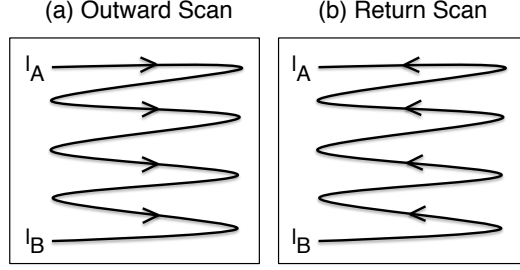


Fig. 4.1: VMN scans the region  $R$

implementation of the time-limited neighbor detector. Note that we do not provide an implementation for the VMN, however, we assume that it can be implemented by the MPE algorithm sketched in Section 4.4.1.

The VMN communicates with the nodes in the network using the timely scoped broadcast service where the broadcast radius equals to a constant  $r_{com}$  known to all nodes. This constant is, in fact, defined by the VMN implementation (see [10]).

The movement of the VMN is defined by a predetermined trajectory function  $loc_{VMN}$  which maps every  $t$  in  $T$  to a location in region  $R$ . This function is known to all nodes in the network. According to  $loc_{VMN}$ , the VMN continuously scans the region  $R$ . The scans are arranged in the form of outward-returns. More precisely, let  $l_A, l_B$  be two distinct locations in  $R$ , then an outward scan starts at  $l_A$  and ends at  $l_B$  and a return scan starts at  $l_B$  and ends at  $l_A$  (see Fig. 4.1). Outward and return scans alternate and the VMN uses exactly the same path in the outward and the return scans. The amount of time that the VMN spends in the outward scan is equal to the amount of time that it spends in the return scan. This time duration is denoted by  $\Delta_{scan}$ .

In order for a scan to cover the entire region (and thus be useful for our time-limited neighbor detector algorithm), the trajectory function of the VMN ensures the following property.

**Scan Completeness.** Let  $s$  be a scan (outward or return) and let  $t_{start}$  be the time when  $s$  is started, then the path traversed by the VMN during  $s$  is such that  $\forall location \in R, \exists t \in [t_{start}; t_{start} + \Delta_{scan}[$  such that  $distance(loc_{VMN}(t), location) \leq r_{com}$ .

### 4.4.3 A Time-limited Neighbor Detector Algorithm

The basic idea behind the algorithm is as follows: time is divided into rounds of duration  $\Delta_{scan}$ . At each round  $k$ , the VMN scans the entire network, collects the predicted locations maps of all nodes, and also sends to each node the neighbors map of that node. The neighbors map is, in fact, generated by the VMN based on the predicted locations maps that are collected at round  $k - 1$ . Note, however, that in the first round the VMN only collects the predicted locations maps and does not send any neighbors maps to the nodes.

The algorithm includes two parts: (1) a part that is executed on each real node  $p_i$  in the network (Algorithm 4.1); (2) a part that is executed on the VMN (Algorithm 4.2). In the following, we discuss the algorithm in more detail.

Since the trajectory function of the VMN is globally known, each node  $p_i$  knows the value of  $\Delta_{scan}$  and can determine when a new round begins (line 7). It can also calculate its distance to the VMN at any time. Thus, at each round  $p_i$  waits until its distance to the VMN becomes less than or equal to  $r_{com}$ . Then, if it has not already sent a message to the VMN in that round, it creates a message  $msg$  to send to the VMN (lines 9-11). This message encapsulates some parameters. Among these parameters the hash map  $locs$  (also referred as the predicted locations map) is used to store the output of PREDICTLOCATIONS primitive of the mobility predictor service (line 12). The parameters  $t_{start}$  and  $t_{end}$  of  $msg$  store, respectively, the beginning and the end of the time interval for which the locations are predicted (lines 13-14).<sup>2</sup> In what follows, for simplicity's sake, we refer to the time interval  $[msg.t_{start}; msg.t_{end}]$  as the *epoch of msg* or  $E_{msg}$ . Once all parameters of  $msg$  are assigned their values,  $msg$  is broadcast within the radius  $r_{com}$ , so it can be received by the VMN (line 16).

When the VMN receives  $msg$  sent by  $p_i$ , it adds  $msg$  to *currentRoundCollectedMsgs* set (lines 31-32). If the VMN is not in the first round and if it has collected a message from  $p_i$  in the previous round, then it creates a message  $VMNmsg$  to reply to  $p_i$  (lines 33-34). The message  $VMNmsg$  encapsulates a hash map  $neighbors$  (also referred as the neighbors map) and some other parameters. To obtain  $neighbors$ , first  $pmsg$  or the message collected from  $p_i$  in the previous round is found (line 35). Then, the function GETNEIGHBORS is called (line 36), which uses the parameters of  $pmsg$  and messages collected from other nodes in the previous round to find neighbors of  $p_i$  during  $E_{pmsg}$  (lines 46-52). Thus,  $neighbors$  contains the set of detected neighbors

---

<sup>2</sup> We assume no clock tick between lines 12-13.

---

**Algorithm 4.1** Time-limited Neighbor Detector Algorithm at Process  $p_i$ 

---

```
1: initialisation:
2:    $noMsgSentInThisRound \leftarrow true$ 
3:   for all  $t \in T$  do
4:      $N(t) \leftarrow \perp$ 

5: PRESENT( $t$ )
6:   return  $N(t)$ 

7: every  $\Delta_{scan}$  do { $p_i$  knows the function  $loc_{VMN}$  and can calculate  $\Delta_{scan}$ }
8:    $noMsgSentInThisRound \leftarrow true$ 

9: upon DISTANCE(GETCURRENTLOCATION,  $loc_{VMN}$ (GETCURRENTTIME))  $\leq r_{com}$  do
10:  if  $noMsgSentInThisRound$  then {a message  $msg$  is created to encapsulate some parameters}
11:     $msg \leftarrow \perp$ 
12:     $msg.locs \leftarrow$  PREDICTLOCATIONS
13:     $msg.t_{start} \leftarrow$  GETCURRENTTIME
14:     $msg.t_{end} \leftarrow msg.t_{start} + \Delta_{predict}$ 
15:     $msg.sender \leftarrow p_i$ 
16:    trigger BROADCAST( $msg, r_{com}$ )
17:     $noMsgSentInThisRound \leftarrow false$ 

18: upon RECEIVE( $VMNmsg, VMN$ ) do
19:  if  $VMNmsg.destination = p_i$  then
20:    for all  $t \in [VMNmsg.t_{start}, VMNmsg.t_{end}]$  do
21:      if  $N(t) = \perp$  then
22:         $N(t) \leftarrow VMNmsg.neighbors(t)$ 
```

---

---

**Algorithm 4.2** Time-limited Neighbor Detector Algorithm at the VMN

---

```
23: initialisation: { $k$  stores the round number}
24:    $k \leftarrow 1$ 
25:    $currentRoundCollectedMsgs \leftarrow \emptyset$ 
26:    $previousRoundCollectedMsgs \leftarrow \emptyset$ 

27: every  $\Delta_{scan}$  do {the VMN knows the function  $loc_{VMN}$  and can calculate  $\Delta_{scan}$ }
28:    $k \leftarrow k + 1$ 
29:    $previousRoundCollectedMsgs \leftarrow currentRoundCollectedMsgs$ 
30:    $currentRoundCollectedMsgs \leftarrow \emptyset$ 

31: upon RECEIVE( $msg, p_i$ ) do
32:    $currentRoundCollectedMsgs \leftarrow currentRoundCollectedMsgs \cup \{msg\}$ 
33:   if  $k > 1 \wedge$  GETPREVIOUSMSG( $p_i$ )  $\neq \perp$  then
34:      $VMNmsg \leftarrow \perp$ 
35:      $pmsg \leftarrow$  GETPREVIOUSMSG( $p_i$ )
36:      $VMNmsg.neighbors \leftarrow$  GETNEIGHBORS( $pmsg$ )
37:      $VMNmsg.t_{start} \leftarrow pmsg.t_{start}$ 
38:      $VMNmsg.t_{end} \leftarrow pmsg.t_{end}$ 
39:      $VMNmsg.destination \leftarrow p_i$ 
40:     trigger BROADCAST( $VMNmsg, r_{com}$ )

41: function GETPREVIOUSMSG( $p_i$ )
42:  for all  $m \in previousRoundCollectedMsgs$  do
43:    if  $m.sender = p_i$  then
44:      return  $m$ 
45:  return  $\perp$ 

46: function GETNEIGHBORS( $pmsg$ )
47:  for all  $t \in [pmsg.t_{start}, pmsg.t_{end}]$  do
48:     $neighbors(t) \leftarrow \emptyset$ 
49:    for all  $m \in previousRoundCollectedMsgs$  do
50:      if  $m.sender \neq pmsg.sender \wedge$  DISTANCE( $m.locs(t), pmsg.locs(t)$ )  $\leq r_d$  then
51:         $neighbors(t) \leftarrow neighbors(t) \cup \{m.sender\}$ 
52:  return  $neighbors$ 
```

---

of  $p_i$  at each time  $t$  during  $E_{pmsg}$ . The parameters  $t_{start}$  and  $t_{end}$  of  $VMNmsg$  store, respectively, the values of  $pmsg.t_{start}$  and  $pmsg.t_{end}$  (lines 37-38). Hence,  $E_{VMNmsg}$  is equal to  $E_{pmsg}$ . Finally,  $p_i$  is specified as the destination of  $VMNmsg$  and it is broadcast within the radius  $r_{com}$  so it can be received by  $p_i$  (lines 39-40). The process  $p_i$  has a hash map  $N$  that is used to store the set of its detected neighbors at any time  $t$  in  $T$ . Thus, when  $p_i$  receives a  $VMNmsg$  that is addressed to it, it looks through  $E_{VMNmsg}$  to find a time  $t$  at which  $N(t)$  is undefined i.e., equals to  $\perp$  (lines 18-21). Then, it assigns  $VMNmsg.neighbors(t)$  to  $N(t)$  (line 22).

#### 4.4.4 Proof of Correctness

In this section we prove that the time-limited neighbor detector algorithm correctly implements the time-limited neighbor detector abstraction under certain conditions. In order to do so, we prove hereafter that the algorithm guarantees the properties of the time-limited neighbor detector abstraction defined in Section 4.3.1.2.

Note that in the following we use the notation  $t_{s,k}$  to refer to the beginning time of a round  $k$  in the algorithm. Thus, for instance  $t_{s,3}$  and  $t_{s,1}$  denote, respectively, the beginning of round 3 and round 1 in the algorithm.

**Theorem 4.1.** *The time-limited neighbor detector algorithm satisfies the time-limited completeness property if the following conditions hold:*

- (a)  $\forall t \in T$ , at least one correct node resides in the circular region of radius  $r_{mp}$  around  $loc_{VMN}(t)$ . This condition, in fact, guarantees that the VMN is correct.
- (b) In each round, the time elapsed from the sending of the predicted locations map to the VMN by a process  $p_i$  until the reception at  $p_i$  of the neighbors map sent by the VMN, is negligible.
- (c)  $\Delta_{predict} \geq \Delta_{future} + 3 \times \Delta_{scan} - 2$ .
- (d) Let  $PRESENT(t)$ , then  $t_c \geq t_{s,3}$ . Recall that  $t_c$  is the time when  $PRESENT(t)$  is invoked. Thus, this condition guarantees that  $PRESENT(t)$  is called in a round  $k$  such that  $k \geq 3$ .
- (e) Let  $PRESENT(t)$ , then  $t \geq t_{s,1} + \Delta_{scan} - 1$ .

*Proof.* We show that with the worst case scenario the algorithm satisfies the *time-limited completeness* property if Conditions (a), (b), (c), (d), (e) hold.

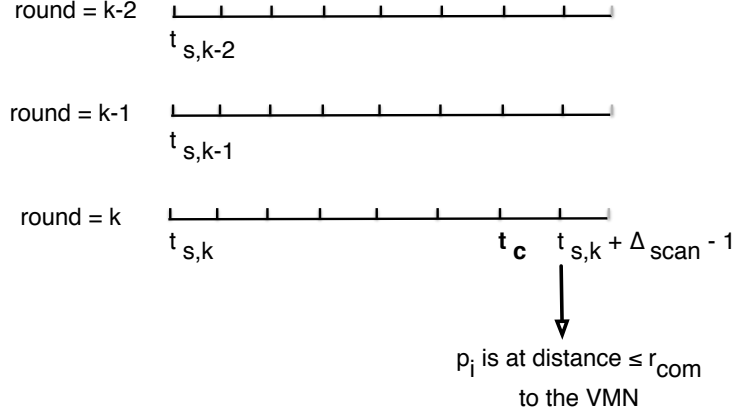


Fig. 4.2: The worst case scenario

Let  $\text{PRESENT}(t)$  be invoked at process  $p_i$  in a round  $k$  i.e.,  $t_c$  be in round  $k$ . We consider the worst case scenario according to which in round  $k$  the distance between  $p_i$  and the VMN becomes less than or equal to  $r_{com}$  only at the last clock tick of the round i.e., at  $t_{s,k} + \Delta_{scan} - 1$  and  $t_c = t_{s,k} + \Delta_{scan} - 2$  (see Fig. 4.2). In addition, for this scenario we assume that  $\Delta_{predict} = \Delta_{future} + 3 \times \Delta_{scan} - 2$ . Thus, we show the proof in the two following cases: (1) for  $t_c \leq t \leq t_c + \Delta_{future}$ ; and (2) for  $t_{s,1} + \Delta_{scan} - 1 \leq t < t_c$ .

1. **For  $t_c \leq t \leq t_c + \Delta_{future}$ .** In the described scenario, the neighbors map sent by the VMN at round  $k$  cannot be used by  $p_i$  for the neighbor detection at time  $t_c$ . The reason is that this neighbors map is only received by  $p_i$  at  $t_{s,k} + \Delta_{scan} - 1$ , i.e., after  $t_c$ . According to Condition (d),  $k$  is at least equal to 3. Thereby, at time  $t_c$ , the process  $p_i$  can rely on the neighbors map that it has received in round  $k - 1$  for neighbor detection.<sup>3</sup> According to the algorithm, the neighbors map that  $p_i$  has received in round  $k - 1$  is made based on the predicted locations maps collected at round  $k - 2$ .

Since  $\Delta_{predict} = \Delta_{future} + 3 \times \Delta_{scan} - 2$ , predicted locations maps of  $p_i$  and  $p_j$  collected at round  $k - 2$  contain the predicted locations for time interval  $[t_c; t_c + \Delta_{future}]$ . In fact, according to the *scan completeness* property of the VMN scans, in a round, for each node there exists a time when its distance to the VMN becomes less than or equal to  $r_{com}$ . According to the algorithm, as soon as a node realizes that it is within distance  $r_{com}$  to the VMN, it sends

<sup>3</sup> Note that according to the algorithm, no neighbors map is distributed by the VMN in round 1.

its predicted locations map to the VMN. Thus, in round  $k - 2$  both  $p_i$  and  $p_j$  send their predicted locations maps to the VMN. Moreover, in round  $k - 2$ , if a node is within distance  $r_{com}$  to the VMN at the earliest possible time (i.e., at time  $t_{s,k-2}$  which is the beginning of the round), its predicted locations map is defined for time interval  $[t_{s,k-2}; t_{s,k-2} + \Delta_{predict}] = [t_{s,k-2}; t_c + \Delta_{future}]$ . Therefore, regardless of the time when  $p_i$  and  $p_j$  are within distance  $r_{com}$  to the VMN, their predicted locations maps in round  $k - 2$  contain predictions for time interval  $[t_c; t_c + \Delta_{future}]$ . Considering Conditions (a) and (b), plus the *timely delivery* property of the underlying timely scoped broadcast, the communication between the VMN and a node is reliable and with negligible delay. Therefore, the messages sent by  $p_i$  and  $p_j$  in round  $k - 2$  are received by the VMN in round  $k - 2$ .

The *scan completeness* property of the VMN scans guarantees that the VMN would be within  $r_{com}$  of  $p_i$  at some time in round  $k - 1$ . At this time, the VMN sends to  $p_i$  its neighbors map which is made based on the predicted locations maps collected at round  $k - 2$ . The *strong accuracy* property of the mobility predictor service guarantees that the location predictions of  $p_i$  and  $p_j$  in their corresponding predicted locations maps are accurate. The function GETNEIGHBORS (lines 46-52) also guarantees the correct neighbor matching between nodes. According to the algorithm, the neighbors map sent to  $p_i$  at round  $k - 1$  is defined for the same time interval as the predicted locations map collected from  $p_i$  at round  $k - 2$ . This means that the neighbors map has the information for time interval  $[t_c; t_c + \Delta_{future}]$ . Thus, based on the arguments above, if  $loc_j(t) \in Z_i(r_d, t)$ ,  $p_j$  is indicated as a neighbor of  $p_i$  at time  $t$  in the neighbors map sent to  $p_i$  in round  $k - 1$ . Again Conditions (a), (b) and the *timely delivery* property of the timely scoped broadcast guarantee that the neighbors map sent to  $p_i$  by the VMN in round  $k - 1$  is received by  $p_i$  in round  $k - 1$ . Then, according to the algorithm, the values of the neighbors map for time interval  $[t_c; t_c + \Delta_{future}]$  are assigned to  $N$  at  $p_i$  (line 22). Hence, if  $loc_j(t) \in Z_i(r_d, t)$ , then  $p_j \in N_i(t)$ .

2. **For  $t_{s,1} + \Delta_{scan} - 1 \leq t < t_c$ .** We prove this case by induction.

*Basis.* We start with the smallest possible value for  $k$ , which according to Condition (d) is  $k = 3$ . If  $k = 3$ ,  $p_i$  can only rely on the neighbors map that it has received in round 2 for neighbor detection. According to the algorithm the neighbors map sent to  $p_i$  at round 2 is defined for the same time interval as the predicted locations map collected from  $p_i$  at round 1. So, even if at round 1,  $p_i$  is within distance  $r_{com}$  to the VMN at the latest possible time (i.e., at

time  $t_{s,1} + \Delta_{scan} - 1$  which is the end of the round), its neighbors map in round 2 is defined for time interval  $[t_{s,1} + \Delta_{scan} - 1; t_{s,1} + \Delta_{scan} - 1 + \Delta_{predict}] = [t_{s,1} + \Delta_{scan} - 1; t_{s,1} + 4 \times \Delta_{scan} + \Delta_{future} - 3]$ . Thus, the neighbors map contains the neighborhood information for time interval  $[t_{s,1} + \Delta_{scan} - 1; t_c[$  and by the same reasoning that in the case (1) above, we conclude that when  $k = 3$ , the *time-limited completeness* is guaranteed for  $t_{s,1} + \Delta_{scan} - 1 \leq t < t_c$ .

*Inductive Step.* We consider some round  $k > 3$ . Then, we show that if the *time-limited completeness* is guaranteed for  $t_{s,1} + \Delta_{scan} - 1 \leq t < t_c$  in round  $k$ , then it is also guaranteed in round  $k + 1$ . From the case (1) above, we know that in round  $k$  the *time-limited completeness* is guaranteed for  $t = t_c$ . Considering this fact and the induction hypothesis, we can say that in round  $k$ , the *time-limited completeness* is guaranteed for  $t_{s,1} + \Delta_{scan} - 1 \leq t \leq t_c$  which is equivalent to say that it is guaranteed for  $t_{s,1} + \Delta_{scan} - 1 \leq t \leq t_{s,k} + \Delta_{scan} - 2$  by replacing  $t_c$  with  $t_{s,k} + \Delta_{scan} - 2$ . Thus, since the algorithm continuously stores the neighborhood information (lines 21-22), in round  $k + 1$  the *time-limited completeness* is already guaranteed for  $t_{s,1} + \Delta_{scan} - 1 \leq t \leq t_{s,k} + \Delta_{scan} - 2$ . Also, we know that in round  $k$ ,  $p_i$  receives a new neighbors map at time  $t_{s,k} + \Delta_{scan} - 1$ . This new neighbors map is made based on the predicted locations map collected at round  $k - 1$  and in the worst case contains the neighborhood information for the time interval  $[t_{s,k-1}; t_{s,k-1} + \Delta_{predict}]$ . The time interval  $[t_{s,k-1}; t_{s,k-1} + \Delta_{predict}]$  can be written as  $[t_{s,k-1}; t_c + \Delta_{future}]$  where  $t_c$  is the  $t_c$  of round  $k + 1$ . Thus, the neighbors map contains the neighborhood information for time interval  $[t_{s,k} + \Delta_{scan} - 1; t_c[$ . Thereby, by the same reasoning that in the case (1) above, we know that in round  $k + 1$ , the *time-limited completeness* is also guaranteed for  $t_{s,k} + \Delta_{scan} - 1 \leq t < t_c$ . Finally, since we showed that in round  $k + 1$ , the *time-limited completeness* is guaranteed for  $t_{s,1} + \Delta_{scan} - 1 \leq t \leq t_{s,k} + \Delta_{scan} - 2$  and for  $t_{s,k} + \Delta_{scan} - 1 \leq t < t_c$ , we conclude that it is guaranteed for  $t_{s,1} + \Delta_{scan} - 1 \leq t < t_c$ .

□

**Theorem 4.2.** *The time-limited neighbor detector algorithm satisfies the perfect accuracy property.*

*Proof.* According to the algorithm, if  $p_j \in N_i(t)$ , there exists a round during which  $p_i$  has received a *VMNmsg* with a *neighbors* map parameter such that  $p_j \in VMNmsg.neighbors(t)$ . Since  $p_i$  verifies the destination of each *VMNmsg* that it receives (line 19), we know that  $p_i$  ignores a *VMNmsgs* that is not addressed to

it. Also, according to the *no creation* property of the timely scoped broadcast service, we know that the  $VMNmsg$  is in fact broadcast by the VMN. According to the algorithm, the *neighbors* map of the  $VMNmsg$  is created by calling the function GETNEIGHBORS (lines 46-52). This function performs the neighbor matching based on *locs* (or the predicted locations maps) parameter of *msg* messages collected from nodes in the previous round. The parameter *sender* of each *msg* guarantees that there is no error regarding the sender of a *msg*. The *strong accuracy* property of the mobility predictor service guarantees that the location predictions in *locs* are accurate. In addition, the function GETNEIGHBORS only detects  $p_j$  as a neighbor of  $p_i$  at time  $t$  if the distance between their predicted locations at  $t$  is less than or equal to  $r_d$  (lines 50-51). Finally, the *no creation* property of the timely scoped broadcast service guarantees that each *msg* collected by the VMN from a node is indeed sent by that node. Therefore if  $p_j \in N_i(t)$ , then  $loc_j(t) \in Z_i(r_d, t)$ .  $\square$

**Theorem 4.3.** *The time-limited neighbor detector algorithm correctly implements the time-limited neighbor detector abstraction if the following conditions hold:*

- (a)  $\forall t \in T$ , at least one correct node resides in the circular region of radius  $r_{mp}$  around  $loc_{VMN}(t)$ . This condition, in fact, guarantees that the VMN is correct.
- (b) In each round, the time elapsed from the sending of the predicted locations map to the VMN by a process  $p_i$  until the reception at  $p_i$  of the neighbors map sent by the VMN, is negligible.
- (c)  $\Delta_{predict} \geq \Delta_{future} + 3 \times \Delta_{scan} - 2$ .
- (d) Let  $PRESENT(t)$ , then  $t_c \geq t_{s,3}$ . Recall that  $t_c$  is the time when  $PRESENT(t)$  is invoked. Thus, this condition guarantees that  $PRESENT(t)$  is called in a round  $k$  such that  $k \geq 3$ .
- (e) Let  $PRESENT(t)$ , then  $t \geq t_{s,1} + \Delta_{scan} - 1$ .

*Proof.* According to Theorem 4.1, the algorithm guarantees the *time-limited completeness* property of the time-limited neighbor detector if Conditions (a), (b), (c), (d), (e) hold. In addition, according to Theorem 4.2, the algorithm guarantees the *perfect accuracy* property of the time-limited neighbor detector. Therefore, the algorithm correctly implements the time-limited neighbor detector abstraction if Conditions (a), (b), (c), (d), (e) hold.  $\square$



## 4.5 Related Work

Neighbor detection in ad hoc networks is usually studied as a building block for applications such as routing, leader election, group management and localization. Many of the existing neighbor detection algorithms belong to the *hello protocols* family [22; 24; 13; 17; 3; 16; 15; 6]. They are based on the *basic hello protocol* first described in *Open Shortest Path First (OSPF)* routing protocol [23]. It works as follows: nodes periodically send *hello* messages to announce their presence to close nodes, and maintain a neighbor set. The sending frequency is denoted by  $f_{hello}$ . If a *hello* message is not received from a neighbor for a predefined amount of time, then that neighbor is discarded from the neighbor set. The problem with this approach is that if  $f_{hello}$  is too low (with respect to the speed of the nodes), then the neighbor set becomes quickly obsolete. On the other hand, if it is too high, the neighbor set remains up to date but it causes a significant waste of communication bandwidth and energy [16]. However, finding the optimal  $f_{hello}$  is not obvious and the existing solutions cannot ideally solve this problem. Moreover, the *hello protocols* usually provide only the set of current neighbors and they do not satisfy any formal guarantees.

Nevertheless, in the literature there exist schemes that use different approaches than the *hello* broadcast for neighbor detection [8; 9]. For instance, in [9], a reliable neighbor detection abstraction is defined that establishes links over which message delivery is guaranteed. The authors present two region-based neighbor detection algorithms which implement the abstraction with different link establishment guarantees. The algorithms are implemented on top of a Medium Access Control (MAC) layer which provides upper bounds on the time for message delivery. The main idea behind the first algorithm is that a node sends a *join* message some time after entering a new region to establish communication links. It also sends a *leave* message some time before leaving a region to inform the other nodes so that they can tear down their corresponding link with that node. To guarantee that these notification messages reach their destination despite the continuous motion of nodes, the authors define the time limits for a node to send the *join* and the *leave* messages. These time limits are obtained using the timing guarantees of the underlying MAC layer. Since a node should send a *leave* message some time before it actually leaves a region, the algorithm assumes that a node's trajectory function is known to that node with enough anticipation to communicate with other nodes before leaving the region. The

first algorithm does not guarantee the communication links when nodes are moving quickly across region boundaries. Thus, the authors introduce a second algorithm. In this new algorithm they apply a technique which overlays multiple region partitions, associating with each region partition an instance of the first algorithm. The output of each instance is then composed such that it guarantees the communication links even when nodes are moving across region boundaries. The approach applied in [9] for neighbor detection is interesting because it uses a relatively lower number of message broadcast compared to the *hello protocols*. Similarly to our work, this approach also uses the knowledge of nodes about their future locations for the neighbor detection. However, contrary to our work, no future neighbor detection is defined and only the current neighbor detection is guaranteed.

We believe that our work is the first attempt to use a virtual node for neighbor detection. The idea of using mobile entities with predefined trajectory to facilitate the design of algorithms for mobile networks was first introduced by Hatzis et al. in [14]. They define the notion of a *compulsory* protocol which requires a subset of the mobile nodes to move in a predefined way. They also present an efficient compulsory protocol for leader election. In [7] and [18] routing protocols for MANETs using compulsory protocols are introduced. In [10; 11], several basic algorithms that use a VMN to solve various problems are briefly presented. These algorithms address the problems such as routing, collecting and evaluating data in mobile ad hoc sensor networks and some other general problems such as group communication and atomic memory.

## 4.6 Conclusion

We have introduced a time-limited neighbor detector service for MANETs. By querying this service a mobile process can know the set of its neighbors at any time in the past, present and up to some bounded time interval in the future. We presented an algorithm that implements this service using a virtual mobile node. Our algorithm is shown to implement correctly the neighbor detector service under certain conditions.

To the best of our knowledge, this is the first work that introduces a neighbor detector service that can detect future neighbors of a node in MANETs. It is also the first paper that uses an approach based on virtual nodes for neighbor detection.

Yet, some issues remain open, which we might consider as future work. For instance, in this work we assumed the *strong accuracy property* for the mobility predictor service. However, in practice such predictions are approximative. Thus, it is interesting to devise an algorithm that can guarantee a certain percentage of correct neighbor detection based on the approximative location predictions. Moreover, one of the conditions for correctness of our algorithm is the correctness of the VMN which is guaranteed if the circular region of radius  $r_{mp}$  around the location of the VMN remains populated all the time. In fact, as also claimed in [11], in the real world this density assumption is reasonable in many cases. For instance, there exist regions (specially in urban areas) that are almost always populated—such as main squares in a downtown area. However, to guarantee the neighbor detection even in less populated regions, we can think of using another type of virtual nodes called *autonomous virtual mobile nodes* [12]. This type of virtual nodes can move autonomously, choosing to change their path based on their own state and inputs from the environment. For instance, if the area in their paths appears deserted, they can change their path to the more populated areas.

Finally, as also claimed in [10], implementing a VMN is expensive. Therefore, it would be interesting to experiment our neighbor detection algorithm with a real implementation to know if the utility outweighs the implementation overhead and possibly to come up with optimizations.

## References

- [1] Apple’s iGroups. <http://tinyurl.com/y9cwvmx>.
- [2] F. Bai and A. Helmy, A survey of mobility modeling and analysis in wireless ad hoc networks, Book Chapter In *Wireless ad hoc and sensor networks*, Springer, 2006.
- [3] M. Bakht, M. Trower, and R. Kravets. Searchlight: helping mobile devices find their neighbors. In *ACM SIGOPS Oper. Syst.*, vol. 45, no. 3, pp. 71–76, 2012.
- [4] Best iPhone bluetooth games. <http://iphone.mob.org/genre/multipleer/>
- [5] B. Bostanipour, B. Garbinato and A. Holzer, Spotcast – A communication abstraction for proximity-based mobile applications, In *Proc. IEEE NCA’12*, pp. 121–129, 2012.

- [6] B. Bostanipour and B. Garbinato, Improving neighbor detection for proximity-based mobile applications, In *Proc. IEEE NCA '13*, pp. 177–182, 2013.
- [7] I. Chatzigiannakis, S. E. Nikolettseas and P. G. Spirakis: An Efficient Communication Strategy for Ad-hoc Mobile Networks. In *Proc. DISC'01*: pp. 285-299, 2001.
- [8] A. Cornejo, S. Viqar, J. L. Welch, Reliable neighbor discovery for mobile ad hoc networks. In *Proc. DIALM-PODC'10*, pp. 63-72, 2010.
- [9] A. Cornejo, S. Viqar and J. L. Welch, Reliable neighbor discovery for mobile ad hoc networks. In *Ad Hoc Networks*. 12 (January 2014), pp. 259–277, 2014.
- [10] S. Dolev, S. Gilbert, N. A. Lynch, E. Schiller, A. A. Shvartsman, J. L. Welch: Virtual Mobile Nodes for Mobile Ad Hoc Networks. In *Proc. DISC'04*, pp. 230–244, 2004.
- [11] S. Dolev, S. Gilbert, N. A. Lynch, E. Schiller, A. A. Shvartsman, J. L. Welch. Virtual mobile nodes for mobile ad hoc networks. Tech Report LCS-TR-937, MIT, 2004.
- [12] S. Dolev, S. Gilbert, E. Schiller, A. A. Shvartsman, J. L. Welch: Autonomous virtual mobile nodes. In *DIALM-POMC*, pp. 62-69, 2005.
- [13] P. Dutta, D. Culler, Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proc. SenSys'08*, 2008.
- [14] K. P. Hatzis, G. P. Pentaris, P. G. Spirakis, V. T. Tampakas and R. B. Tan. Fundamental control algorithms in mobile networks. In *Proc. ACM SPAA'99*, 1999.
- [15] D. He, N. Mitton, and D. Simplot-Ryl, An energy efficient adaptive HELLO algorithm for mobile ad hoc networks, In *Proc. ACM MSWiM'13*, pp. 65–72, 2013.
- [16] F. Ingelrest, N. Mitton, and D. Simplot-Ryl, A turnover based adaptive hello protocol for mobile ad hoc and sensor networks, In *Proc. MASCOTS'07*, pp. 9–14, 2007.
- [17] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar, U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol, In *IPSN'10*, pp. 350–361, 2010.
- [18] Q. Li and D. Rus. Sending messages to mobile users in disconnected ad-hoc wireless networks. In *Proc. MobiCom*, 2000.
- [19] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

- [20] LocoPing. <http://www.locoping.com/>
- [21] LoKast. <http://www.lokast.com/>
- [22] M. J. McGlynn and S. A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proc. ACM MobiHoc'01*, pp. 137–145, 2001.
- [23] J. Moy, OSPF – Open Shortest Path First, RFC 1583, 1994.
- [24] G. Wattenhofer, G. Alonso, E. Kranakis, and P. Widmayer. Randomized protocols for node discovery in ad hoc, single broadcast channel networks. In *Proc. IPDPS'03*, 2003.
- [25] Waze. <https://www.waze.com/en/>
- [26] WhosHere. <http://whoshere.net/>



## Chapter 5

# A Neighbor Detection Algorithm Based on Multiple Virtual Mobile Nodes for Mobile Ad Hoc Networks

**Abstract** We introduce an algorithm that implements a time-limited neighbor detector service in mobile ad hoc networks. The time-limited neighbor detector enables a mobile device to detect other nearby devices in the past, present and up to some bounded time interval in the future. In particular, it can be used by a new trend of mobile applications known as proximity-based mobile applications. To implement the time-limited neighbor detector, our algorithm uses  $n = 2^k$  virtual mobile nodes where  $k$  is a non-negative integer. A virtual mobile node is an abstraction that is akin to a mobile node that travels in the network in a predefined trajectory. In practice, it can be implemented by a set of real nodes based on a replicated state machine approach. Our algorithm implements the neighbor detector for real nodes located in a circular region. We also assume that each real node can accurately predict its own locations up to some bounded time interval  $\Delta_{predict}$  in the future. The key idea of the algorithm is that the virtual mobile nodes regularly collect location predictions of real nodes from different subregions, meet to share what they have collected with each other and then distribute the collected location predictions to real nodes. Thus, each real node can use the distributed location predictions for neighbor detection. We show that our algorithm is correct in periodically well-populated regions. We also define the minimum value of  $\Delta_{predict}$  for which the algorithm is correct. Compared to the previously proposed solution also based on the notion of virtual mobile nodes, our algorithm has two advantages: (1) it tolerates the failure of one to all virtual mobile nodes; (2) as  $n$  grows, it remains correct with smaller values of  $\Delta_{predict}$ . This feature makes the real-world deployment of the neighbor detector easier since with the existing prediction methods, location predictions usually tend to become less accurate as  $\Delta_{predict}$  increases. We also show that the cost of our algorithm (in terms of communication) scales linearly with the number of virtual mobile nodes.

## Publication:

B. Bostanipour and B. Garbinato, A neighbor detection algorithm based on multiple virtual mobile nodes for mobile ad hoc networks, *currently under minor revision for publication in Elsevier Computer Networks Journal*, 2016.

The primary version of this work is published as a conference paper:

B. Bostanipour and B. Garbinato, Neighbor detection based on multiple virtual mobile nodes, *In Proceedings of the 24th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'16)*, pp. 322–327, Heraklion, Greece, IEEE, 2016.

## 5.1 Introduction

The growing adoption and usage of mobile devices and particularly smartphones has caused the emergence of a new trend of distributed applications known as *Proximity-Based Mobile (PBM)* applications [3; 4; 5; 6]. These applications enable a user to interact with others in a defined range and for a given time duration e.g., for social networking (WhosHere [52], LoKast [35], iGroups [25], LocoPing [34]), gaming (local multiplayer apps [36]) and driving (Waze [51]).

Discovering who is nearby is one of the basic requirements of PBM applications. It is the preliminary step for further interactions between users. It also enables users to extend their social network from the people that they know to the people that they might not know but who are in their proximity. For instance, with the social networking applications such as WhosHere [52] or LoKast [35], a user first discovers others in her proximity and then decides to view their profiles, start a chat with them or add them as friends. The discoverability, however, may not always be limited to the current neighbors. For instance, with the social networking applications such as iGroups [25] or LocoPing [34], a user can discover others who were in her vicinity during a past event (e.g., concert, tradeshow, wedding) or simply during a past time interval (e.g., the past 24 hours). One can also think of applications that provide the user with the list of people who will be in her proximity up to some time interval in the future and thus create the potential for new types of social interactions [5].

In this paper, we present an algorithm that implements the *time-limited neighbor detector* service. This service enables a device to discover the set of its neighbors in



the past, present and up to some bounded time interval in the future in a mobile ad hoc network (MANET). It was first introduced in our previous work [5] to capture the requirements of neighbor detection in PBM applications.

Our algorithm implements the time-limited neighbor detector using  $n = 2^k$  virtual mobile nodes where  $k$  is a non-negative integer. A virtual mobile node is an abstraction that was already introduced in the literature and used for tasks such as routing or collecting data in MANETs [14; 15]. It is akin to a mobile node that travels in the network in a predefined trajectory known in advance to all nodes. In practice a virtual mobile node is emulated by a set of real nodes in the network using a replicated state machine approach.

Our algorithm implements the neighbor detector for real nodes located in a circular region. We also assume that each real node can accurately predict its own locations up to some bounded time interval  $\Delta_{predict}$  in the future. Thus, the region is divided into  $n$  equal subregions and each subregion is associated with one virtual mobile node. Each virtual mobile node regularly collects the location predictions from the real nodes in its subregion and meets other virtual mobile nodes to share what it has collected with them. After the sharing, each virtual mobile node has the location predictions collected from the entire region, which it distributes to the real nodes in its subregion. In this way, each real node can find its neighbors at current and future times based on the collected location predictions that it receives from a virtual mobile node. It can also store the collected location predictions so it can be queried about its past neighbors.

**Main Contributions.** The main contributions of this paper are as follows. We introduce an algorithm that implements the time-limited neighbor detector service using  $n = 2^k$  virtual mobile nodes where  $k$  is a non-negative integer. To guarantee the coordination between the virtual mobile nodes, we define a set of explicit properties for their trajectory functions and we show how such trajectory functions can be computed. We prove the correctness of the algorithm under certain conditions. In particular, we show that our algorithm is correct for a category of executions, called *nice executions*, which basically correspond to the executions of the algorithm in periodically well-populated regions such as main squares in a downtown area. We also define the minimum value of  $\Delta_{predict}$  for which the algorithm is correct in different cases of nice executions.

This work relies on our previous work [5] for the general idea of using virtual mobile nodes and location predictions to implement the time-limited neighbor detector.

However, contrary to the algorithm in [5] which uses only a single virtual mobile node and does not tolerate its failure, our algorithm can use multiple virtual mobile nodes and can tolerate the failure of one to all virtual mobile nodes. Due to the use of multiple virtual mobile nodes, our algorithm has a feature which did not exist in the previous solution: as the number of virtual mobile nodes grows, our algorithm remains correct with smaller values of  $\Delta_{predict}$ . This feature makes the real-world deployment of the neighbor detector easier. In fact, although there exist different approaches to predict the future locations of a real node, usually predictions tend to become less accurate as  $\Delta_{predict}$  increases. We show that the cost of our algorithm (in terms of communication) scales linearly with the number of virtual mobile nodes. We also propose a set of optimizations which can be used for the real-world deployment of our algorithm.

To the best of our knowledge, this is the first work that uses multiple virtual mobile nodes to implement a neighbor detector service in MANETs. Moreover, this is the first work that defines a set of explicit properties for the trajectory functions of the virtual mobile nodes to guarantee the coordination between them.

**Road Map.** The remainder of the paper is as follows. In Section 5.2, we describe our system model and introduce some definitions. In Section 5.3, we present a neighbor detector service for MANETs in two variants: the *perfect* variant, which corresponds to the ideal case of neighbor detection and is rather impractical and the *time-limited* variant, for which we propose an implementation in this paper. In Section 5.4, we present the implementation of the time-limited variant of the neighbor detector service based on virtual mobile nodes. In order to do so, we first describe what a virtual mobile node is and how it can be used for the implementation of the time-limited neighbor detector. We then add  $n$  virtual mobile nodes to the system model. Each virtual mobile node has a so called *scan path* through which it travels in its subregion. Thus, we define the properties of this path and we show how it can be computed in order to be useful for our algorithm. We then introduce a round-based algorithm that implements the time-limited neighbor detector in the new system model and prove the correctness of the algorithm under certain conditions. As we show in the proof, the algorithm can tolerate the failure of one to all virtual mobile nodes for a category of executions, called *nice executions*, which basically correspond to the executions of the algorithm in periodically well-populated regions. We also define the minimum value of  $\Delta_{predict}$  for which the algorithm is correct in different cases of nice executions. Then, we show the evolution of this value as  $n$  grows.

Based on this evolution, we deduce that as the number of virtual mobile nodes grows the algorithm requires smaller values of  $\Delta_{predict}$  to correctly implement the time-limited neighbor detector. In Section 5.5, we discuss two topics related to the performance of the algorithm, namely its scalability with respect to the number of virtual mobile nodes and the optimizations which can improve its performance. In particular, we show that the communication cost of the algorithm, defined as the number of message broadcasts per round, has a complexity of  $\mathcal{O}(n)$ . In Section 5.6, we discuss the related work and in Section 5.7, we conclude and discuss future work. The paper has also an Appendix 5.A, which is devoted to finding an upper bound for the scan path length of a virtual mobile node. This upper bound is used (directly or indirectly) in Sections 5.4.3, 5.4.6 and 5.5.1 to find other results.

## 5.2 System Model and Definitions

We consider a mobile ad-hoc network (MANET) consisting of a set  $P$  of processes that move in a two dimensional plane. A process abstracts a mobile device in a PBM application.<sup>1</sup> We use the terms *process*, *node* and *real node* interchangeably. Each process has a unique identifier. Processes can move on any continuous path, however there exists a known upper bound on their motion speed. A process is prone to *crash-reboot* failures: it can fail and recover at any time, and when the process recovers, it returns to its initial state. A process is *correct* if it never fails. Since we do not consider *Byzantine* behaviors, the information security and privacy issues are beyond the scope of this paper.

We assume the existence of a discrete global clock, i.e., the range  $T$  of the clock's ticks is the set of non-negative integers. We also assume the existence of a known bound on the relative processing speed. Each process in the system has access to a *LocalCast service*, a *global positioning service* and a *mobility predictor* service. In the following, we first introduce some definitions that are used throughout the

---

<sup>1</sup> There are two main reasons behind our choice of a MANET as the underlying network architecture. Firstly, MANETs seem to be the most natural existing technology to enable PBM applications. In fact, similarly to PBM applications, in a MANET two nodes can communicate if they are within a certain distance of each other (to have radio connectivity) for a certain amount of time. Secondly, mobile devices are increasingly equipped with ad hoc communications capabilities (e.g., WiFi in ad hoc mode or Bluetooth) which increases the chance of MANETs to be one of the future mainstream technologies for PBM applications [6].

paper. We then present each of the above mentioned services and for each service, we describe how it can be implemented in the real world.

### 5.2.1 Definitions

Let  $p_i$  be a process in the network, we introduce the following definitions in order to capture proximity-based semantics.

- A *location* denotes a geometric point in the two dimensional plane and can be expressed as tuple  $(x, y)$ .
- $loc(p_i, t)$  denotes the location occupied by process  $p_i$  at time  $t \in T$ .
- $Z(p_i, r, t)$  denotes all the locations inside or on the circle centered at  $loc(p_i, t)$  with given radius  $r$ .
- $r_d$  is called *the neighbor detection radius*. It is a constant known by all processes in the network. Thus,  $Z(p_i, r_d, t)$  presents the *neighborhood region* of  $p_i$  at time  $t$ .

### 5.2.2 LocalCast Service

This communication service was introduced in [14; 15]. It allows a process to send messages to all processes located within a given radius around it. Formally, the LocalCast service exposes the following primitives:

- $BROADCAST(m, r)$ : broadcasts a message  $m$  in  $Z(p_i, r, t_b)$ , where  $p_i$  is the sender and  $t_b$  is the time when the broadcast is invoked.
- $RECEIVE(m, p_i)$ : callback delivering a message  $m$  broadcast by process  $p_i$ .

The service satisfies the following properties.

**Reliable Delivery.** Assume that a process  $p_i$  performs a  $BROADCAST(m, r)$  action. Let  $d$  be a constant and  $\Delta_{delivery} = [t_b; t_b + d]$ . Then every process  $p_j$  delivers  $m$  in  $\Delta_{delivery}$  if  $\forall t \in \Delta_{delivery}, loc(p_j, t) \in Z(p_i, r, t)$  and  $p_j$  does not fail during  $\Delta_{delivery}$ .

**Integrity.** For any LocalCast message  $m$  and process  $p_i$ , if  $RECEIVE(m, p_j)$  event occurs at  $p_i$ , then a  $BROADCAST(m, r)$  event precedes it at some process  $p_j$ .

As stated in [14], sending a message using this service should be thought of as making a single wireless broadcast (with a small number of retries, if necessary, to avoid collisions). In practice, this service can be implemented with high probability by one of the existing single-hop wireless broadcast protocols as long as the broadcast radius is not too large [14; 15].

### 5.2.3 Global Positioning Service

This service allows each mobile process  $p_i$  to know its current location and the current time via the following functions:

- **GETCURRENTTIME**: returns the current global time. Formally, this implies that each process  $p_i$  has access to the global clock modeled at the beginning of Section 5.2.
- **GETCURRENTLOCATION**: returns the location occupied by  $p_i$  at the current global time.

In this paper, we do not provide any formal properties for this service. However, we assume that the outputs of its functions are accurate. In an outdoor setting, this service can typically be implemented using NASA’s GPS space-based satellite navigation technology. In an indoor environment, a MITs Cricket device [39] may be more suitable to implement this service.

### 5.2.4 Mobility Predictor Service

This service allows each mobile process  $p_i$  to predict its future locations up to some bounded time  $\Delta_{predict}$  via the following function:

- **PREDICTLOCATIONS**: returns a hash map containing the predicted locations for  $p_i$  at each time  $t$  in the interval  $[t_c; t_c + \Delta_{predict}]$  where  $t_c$  is the time when **PREDICTLOCATIONS** is invoked.

The service satisfies the following property.

**Strong Accuracy.** Let  $t \in [t_c; t_c + \Delta_{predict}]$  and  $l$  be a location, if  $p_i$  is predicted to be at  $l$  at time  $t$ , then  $loc(p_i, t) = l$ .

In this paper we assume that in a PBM application a mobile device (abstracted by a process) is used only by one user. Therefore, to implement the mobility predictor service, a human mobility prediction method should be considered. The human mobility prediction methods are usually based on the fact that the human activities are characterized by a certain degree of regularity and predictability [40], thus, a person’s future movement can be predicted using her prior movement history (e.g., previous locations, residence time at each previous location, etc...). In the literature, there exist various human mobility prediction methods [10; 44; 45; 13; 40], which are not all suitable for implementing the mobility predictor service. In fact, from a practical point of view, a prediction method should have the following characteristics to implement our mobility predictor service: (1) it should be able to predict not only the future locations of a user but also the time interval during which the user stays at each predicted location; (2) it should not require complex computations and large amount of memory space. The reason behind this requirement is that we consider a MANET where fixed infrastructures are lacking. Thereby, the predictions should be made by each device itself, which has limited resources in terms of battery life and computational capacity. The Markov-based method introduced in [45] and the method based on nonlinear time series analysis introduced in [40] are examples of the prediction methods that satisfy the above mentioned requirements and can be used to implement our mobility predictor service.

Note that for simplicity’s sake, the mobility predictor service that we consider in this paper is an idealized predictor. In practice, the above mentioned prediction methods can implement it with high probability as long as  $\Delta_{predict}$  is not large (i.e., less than or equal to five minutes). More precisely, the mobility predictor service has a *strong accuracy* property according to which the predictions are 100% accurate through the whole interval  $\Delta_{predict}$ . However, in reality the above mentioned methods can predict the next immediate location with a high accuracy (from 75% to 95% accuracy depending on the method) and this accuracy decreases in an almost linear way as  $\Delta_{predict}$  increases. In general, we can say that with these prediction methods the predictions are still highly accurate for a  $\Delta_{predict}$  equal to five minutes (for more details see the evaluations in [40]). Another characteristic of our mobility predictor service is that it makes the predictions for geometric locations. However, many of the existing mobility prediction methods make predictions for *symbolic* locations (e.g., rooms in a building, special areas in a map, etc...). In fact, since with symbolic locations, identifying a node’s neighbors will depend directly on how symbolic loca-

tions are defined and since there exist different types of symbolic locations, in this paper for simplicity's sake, we assume that the predictions are made for geometric locations. We believe that our neighbor detector algorithm can be adapted to the particular cases where symbolic locations are used.

### 5.3 The Neighbor Detector Service

This service was first introduced in our previous work in [5]. Intuitively, the neighbor detector service allows a process to know its neighbors at a given time. Formally, it exposes the following primitive:

- **PRESENT( $t$ )**: returns  $N(p_i, t)$  i.e., the set of processes detected as neighbors of  $p_i$  at time  $t$ , where  $p_i$  is the process that invokes **PRESENT**.

#### 5.3.1 Neighbor Detector Variants

We present two variants of the neighbor detector service: *the perfect neighbor detector* and *the time-limited neighbor detector*. As we discuss in the following, *the perfect neighbor detector* presents an ideal case of neighbor detection and is rather impractical. The reason why we present this variant is to help the reader to better understand the properties of the other variant i.e., *the time-limited neighbor detector*. *The time-limited neighbor detector* is more practical and is the variant for which we propose an implementation in this paper.

##### 5.3.1.1 Perfect Neighbor Detector

By querying this variant of neighbor detector service, a mobile process is able to know the set of its neighbors at any time in the past, present or the future.

**Perfect Completeness.** Let  $p_i$  and  $p_j$  be two correct processes, if  $loc(p_j, t) \in Z(p_i, r_d, t)$ , then  $p_j \in N(p_i, t)$ .

**Perfect Accuracy.** Let  $p_i$  and  $p_j$  be two correct processes, if  $p_j \in N(p_i, t)$ , then  $loc(p_j, t) \in Z(p_i, r_d, t)$ .

Roughly speaking, the *perfect completeness* property requires a neighbor detector to detect any node that is in the neighborhood region at any time in the past, present or future. At the same time, the *perfect accuracy* property guarantees that no false detection occurs. Since in practice implementing the *perfect completeness* property requires an infinite knowledge of nodes' locations in the future, we consider a more practical variant of the neighbor detector service called *the time-limited neighbor detector*. We introduce this variant hereafter.

### 5.3.1.2 Time-limited Neighbor Detector

Compared to the *perfect neighbor detector*, this variant has a different *completeness* property. However, its *accuracy* property is the same. We define its properties, below.

***Time-limited Completeness.*** Let  $p_i$  and  $p_j$  be two correct processes and  $\Delta_{future}$  be a bounded time interval such that  $\Delta_{future} > 0$ , if  $loc(p_j, t) \in Z(p_i, r_d, t)$  and  $t \leq t_c + \Delta_{future}$ , then  $p_j \in N(p_i, t)$ , where  $t_c$  is the time when PRESENT is invoked at  $p_i$ .

***Perfect Accuracy.*** Let  $p_i$  and  $p_j$  be two correct processes, if  $p_j \in N(p_i, t)$ , then  $loc(p_j, t) \in Z(p_i, r_d, t)$ .

Similarly to the *perfect completeness* property, the *time-limited completeness* property requires a neighbor detector to detect any node that is in the neighborhood region at any time in the past or present. However, its ability to detect future neighbors is limited by a bounded time duration  $\Delta_{future}$ . More precisely, it only detects a node that is in the neighborhood region at any time from the time when PRESENT is invoked up to  $\Delta_{future}$ .<sup>2</sup> The *perfect accuracy* property also guarantees no false detection.

As already stated, in this paper we propose an implementation for the time-limited neighbor detector variant. Thus, henceforth whenever we use the term *the neighbor detector service*, we actually refer to the time-limited neighbor detector.

---

<sup>2</sup> For simplicity's sake, we do not assume a time bound on the availability of the past neighborhood information at this point. We will discuss this further in Section 5.5.2.2.



## 5.4 Implementing The Time-Limited Neighbor Detector

To implement the time-limited neighbor detector, our intuition is as follows: since each node knows its own locations up to  $\Delta_{predict}$  in the future, we can think of a moving entity that travels through the network, collects the location predictions of all nodes, and then distributes all the collected location predictions to the nodes. In this way, each node can find its neighbors at current and future times based on the collected location predictions. It can also store the collected location predictions so it can be queried about its past neighbors. In our solution, we consider a *virtual mobile node* (first introduced in [14]) to be used as the moving entity. Moreover, to simplify the problem, we perform the neighbor detection only for real nodes which are in a circular region  $R$  of the two dimensional plane. However, using only one virtual mobile node to implement the neighbor detector has a main disadvantage: as the size of the region  $R$  grows, the virtual mobile node spends more time to travel through the network. This can cause the collected location predictions to expire before they can be used for neighbor detection. One way to overcome this problem is to increase  $\Delta_{predict}$  of the mobility predictor. However, as discussed in Section 5.2.4, with the existing mobility prediction methods, predictions usually tend to become less accurate as  $\Delta_{predict}$  increases.

Another way to deal with this problem is to decrease the traveling time of the virtual mobile node. In order to do so, our solution consists of using more than one virtual mobile node. In fact, our solution can work with  $n = 2^k$  virtual mobile nodes where  $k$  is a non-negative integer. Thus, the region is divided into  $n$  equal subregions and each subregion is associated with one virtual mobile node. Virtual mobile nodes collect simultaneously the location predictions from the real nodes in their subregions and meet at the center of  $R$  to share what they have collected with each other. After the sharing, every virtual mobile node has the location predictions collected from the entire  $R$ . Then, the virtual mobile nodes simultaneously distribute the collected location predictions to the real nodes in their corresponding subregions. As we further show, as  $n$  grows, our solution can correctly implement the neighbor detector with smaller values of  $\Delta_{predict}$ . Intuitively, this is because as  $n$  grows,  $R$  is divided into more and consequently smaller subregions and each virtual mobile node spends less time to travel through its subregion.

In the following, we first describe what a virtual mobile node is and we add  $n$  virtual mobile nodes to the system model. We also define the properties of the *scan*

*path* i.e., the path through which a virtual mobile node travels in its subregion and we show how it can be computed. We then introduce an algorithm that implements the neighbor detector in the new system model and we prove the correctness of the algorithm. As we show in the proof, the algorithm can tolerate the failure of the virtual mobile nodes under certain conditions. We also define the minimum value of  $\Delta_{predict}$  for which the algorithm is correct. We then show the evolution of this value as  $n$  grows. Based on this evolution, we deduce that as the number of virtual mobile nodes grows the algorithm requires smaller values of  $\Delta_{predict}$  to correctly implement the neighbor detector.

### 5.4.1 Virtual Mobile Node

A virtual mobile node (also referred to as a *virtual node*) is an abstraction that is akin to a mobile node that travels in the network in a predefined trajectory. It was first introduced by Dolev et al. in [14] to simplify the task of designing algorithms for mobile ad hoc networks. In fact, Dolev et al. consider two main reasons behind the difficulty of designing algorithms for mobile ad hoc networks: (1) the movement of a mobile node is unpredictable; (2) mobile nodes are unreliable i.e., they can continuously join and leave the system, they may fail or recover or be turned on and off by the user or may sometimes choose to sleep and save power. Thus, a virtual mobile node is designed so that it can execute any distributed algorithm that a real node can execute, however, its movement can be predefined and known in advance to all real nodes in the network. Moreover, a virtual mobile node is reliable (also called *robust*). Roughly speaking, this means that a virtual mobile node does not fail as long as it travels through well-populated areas of the network [14].

In [14] an algorithm called *Mobile Point Emulator (MPE)* is introduced, which implements the virtual mobile node abstraction in a system model equivalent to the system model defined in this paper. The implementation of the virtual mobile node is based on a replicated state machine technique similar to the one originally presented in [33]. In fact, in order to achieve the robustness of the virtual mobile node in spite of the failure of the real nodes, the algorithm replicates the state of the virtual mobile node at the real nodes which travel near the location of the virtual mobile node. More precisely, the algorithm defines a *mobile point* to be a circular region of a radius  $r_{mp}$ , which moves according to the predefined path of the virtual mobile node, i.e., at time

$t$  the center of the mobile point coincides with the preplanned location of the virtual mobile node at time  $t$ . The MPE replicates the state of the virtual mobile node at every real node within the mobile point’s region, modifying the set of replicas as the real nodes move in and out of the mobile point’s region. MPE uses a total-order broadcast service to ensure that the replicas are updated consistently. The total order broadcast service is built using the LocalCast communication service (defined in Section 5.2.2) and synchronized clocks which are obtained by using a service equivalent to our global positioning service. Note that the real nodes are only used by the MPE algorithm to assist in emulating the virtual mobile node. Thereby, the motion of a virtual mobile node may be completely uncorrelated with the motion of the real nodes i.e., even if all the real nodes are moving in one direction, the virtual mobile node may travel in the opposite direction.

Similarly to a real node, a virtual mobile node can communicate with other virtual or real nodes using the LocalCast service. Also, a virtual mobile node is prone to *crash-reboot* failures. It can crash if and only if its trajectory takes it into a region unpopulated by any real nodes (i.e., where there are no real nodes to act as replicas), however, it recovers to its initial state as soon as it reenters a dense area. A virtual mobile node is *correct* if it never fails, i.e.,  $\forall t \in T$ , at least one correct real node resides in the circular region of radius  $r_{mp}$  around the preplanned location of the virtual mobile node at time  $t$ .

## 5.4.2 Adding Virtual Mobile Nodes to the System Model

In this section, we add a set of  $n$  virtual mobile nodes  $V = \{v_1, \dots, v_n\}$  to the system model where  $n = 2^k$  and  $k$  is a non-negative integer. Each virtual node is assigned a unique identifier. Note that we do not provide an implementation for the virtual nodes, however, we assume that they can be implemented by the MPE algorithm sketched in Section 5.4.1.

Let region  $R$  be a closed disk of radius  $r_{map}$ , centered at location  $l_{map-center}$  which is the origin of the two dimensional plane. Each virtual node  $v_i$  is associated with a subregion  $R_i$  of  $R$ . The subregion  $R_i$  is a sector of  $R$  (in the shape of a pizza slice)

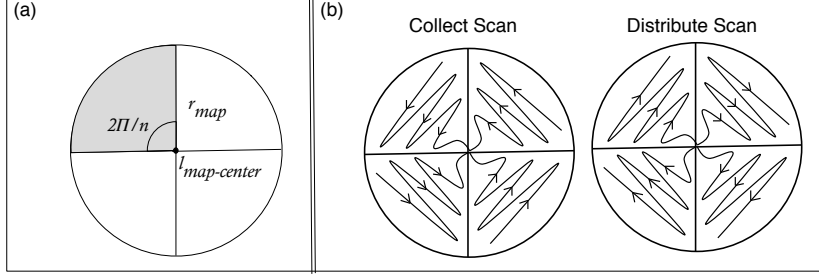


Fig. 5.1: The subfigures correspond to the case where the number of virtual mobile nodes (denoted by  $n$ ) is equal to four. (a) Disk  $R$  is presented where the grey area corresponds to a subregion  $R_i$ . (b) Each virtual mobile node scans its associated sub-region in the form of collect and distribute scans. The arrows indicate the direction of motion.

enclosed by two radii and an arc, where the arc subtends an angle  $\frac{2\pi}{n}$  (See Fig. 5.1.a). All subregions have the same area and  $\bigcup_{i=1}^{i=n} R_i = R$ .<sup>3</sup>

A virtual node can communicate with other virtual nodes or the real nodes using the LocalCast service (defined in Section 5.2.2) where the broadcast radius equals to a constant non-negative integer  $r_{com}$  known globally. This constant is defined by the virtual node implementation (see [14]). Moreover, similar to a real node, a virtual node has access to the global positioning service (defined in Section 5.2.3).

The movement of a virtual node  $v_i$  is defined by a predetermined trajectory function  $loc(v_i, t)$ , which maps every  $t$  in  $T$  to a location. This function is known to all virtual nodes and real nodes in the network. The average speed of  $v_i$ 's movement is equal to a constant  $v_{avg}$ . This constant is defined by the speed of the real nodes (that emulate  $v_i$ ) and the speed of the subprotocols of the MPE algorithm sketched in Section 5.4.1.

The trajectory function of  $v_i$  is defined such that it can be used by our algorithm for the implementation of the neighbor detector. According to the trajectory function,  $v_i$  continuously scans the subregion  $R_i$ . The scans are arranged in the form of collect-distribute. More precisely, let  $l_{init}(v_i)$  be a location different from  $l_{map-center}$ . Then, a collect scan starts at  $l_{init}(v_i)$  and ends at  $l_{map-center}$  and a distribute scan

<sup>3</sup> In general, a disk can be divided using straightedge and compass into  $n$  equal parts if  $n = 2^k m$  where  $k$  is a non-negative integer and  $m$  is either equal to 1 or else  $m$  is a product of different Fermat primes [27].

starts at  $l_{map-center}$  and ends at  $l_{init}(v_i)$  (See Fig. 5.1.b). The first scan starts at time  $t = 0$  and is a collect scan. Collect and distribute scans alternate and  $v_i$  uses exactly the same path in the collect and the distribute scans. This path is called the *scan path* of  $v_i$  and its length is denoted by  $L_{scan-path}(v_i)$ . The amount of time that  $v_i$  spends in a collect scan is equal to the amount of time that it spends in a distribute scan. This time duration is denoted by  $\Delta_{scan}(v_i)$ .

In order to be useful for our neighbor detector algorithm, the scan path of  $v_i$  should satisfy the three following properties:

**Scan Completeness.** Let  $s$  be a scan (collect or distribute) and let  $t_{begin}$  be the time when  $s$  begins, then the path traversed by  $v_i$  during  $s$  is such that  $\forall location \in R_i, \exists t \in [t_{begin}; t_{begin} + \Delta_{scan}(v_i) - 1]$  s.t.  $distance(loc(v_i, t), location) \leq r_{com}$ .

**Equal Scan Path Lengths.** Let  $v_j$  be a virtual node different from  $v_i$ , then  $L_{scan-path}(v_i) = L_{scan-path}(v_j)$ .

**Proportional Scan Path Length.**  $L_{scan-path}(v_i)$  is an inverse function of  $n$ .

The *scan completeness* property guarantees that a scan covers the entire subregion  $R_i$  in terms of  $r_{com}$ . With regard to the *equal scan path lengths* property, it has a direct result, that is, the value of  $\Delta_{scan}$  is the same for all virtual nodes (recall that all virtual nodes have the same average speed  $v_{avg}$ ). Since all virtual nodes start their scanning at  $t = 0$  and with a collect scan, this guarantees that all virtual nodes meet at the end of each collect scan at  $l_{map-center}$ . Finally, *proportional scan path length* guarantees that as  $n$  (i.e., the number of virtual nodes) grows, the scan path length and consequently  $\Delta_{scan}$  of each virtual node decreases.

In next section, we define the scan path that satisfies these properties and is used by the trajectory functions of the virtual nodes.

### 5.4.3 The Scan Path of a Virtual Mobile Node

As described in Section 5.4.2, the scan path of a virtual node  $v_i$  should satisfy a set of properties. We start by finding the optimal path that satisfies the *scan completeness* property. We then show that this path also satisfies the *equal scan path lengths* and the *proportional scan path length* properties.

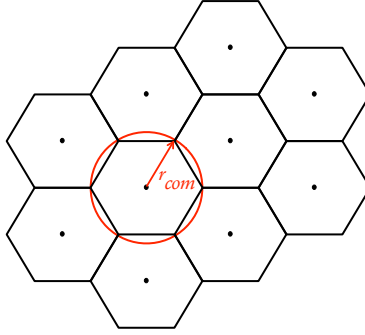


Fig. 5.2: Hexagonal tessellation of the surface of a subregion  $R_i$ . Each hexagon approximates a circle of radius  $r_{com}$  and hence its circumradius is equal to  $r_{com}$ . In the figure, we present the circle and its radius in red only for one hexagon.

The optimal path that satisfies the *scan completeness* property is the shortest possible path that goes through a set of locations that we call *covering centers*. Roughly speaking, the covering centers are such that if  $v_i$  broadcasts a message at all covering centers then the message is disseminated at all locations in  $R_i$ . Thus, covering centers are centers of disks of radius  $r_{com}$  that cover the whole surface of  $R_i$  such that the number of disks is minimum. Note that a part of the surface of some of these disks can be located out of  $R_i$ , thereby, some of the covering centers can be situated at the boundary or even out of  $R_i$ .

Finding covering centers is a NP hard problem [29]. It can be approximately solved by applying *hexagonal tessellation* (or so called *hexagonal tiling*) [21]. More precisely, the surface of  $R_i$  is tessellated using the regular hexagons of circumradius  $r_{com}$  (See Fig. 5.2). Since for all virtual nodes the scan path goes through  $l_{map-center}$ , the tessellation made by the tessellation algorithm is such that one of the hexagons is centered at  $l_{map-center}$ . The algorithm also ensures that the number of hexagons covering  $R_i$  is minimum. Once the tessellation is made, the centers of the hexagons are identified as the covering centers. Thus, the scan path can be found as the shortest possible route that visits the center of each hexagon exactly once. This is a variant of a famous algorithmic problem known as the *Travelling Salesman Problem* (*TSP*). In this case, the problem can be easily solved thanks to the properties of the hexagonal tessellation. In fact, in the hexagonal tessellation, the distance between the centers of any two adjacent hexagons is equal to  $\sqrt{3}r_{com}$ . Therefore, the scan path can be found as the path that connects the centers of each pair of adjacent hexagons exactly once.

The scan path that we have found satisfies the *equal scan path lengths* property since the tessellation of every subregion is made by applying the same algorithm and all subregions have the same shape and area. We would like also to show that the scan path satisfies the *proportional scan path length* property. In fact, the scan path length of  $v_i$  can be found as:

$$L_{scan-path}(v_i) = (NOC(R_i) - 1) \times \sqrt{3}r_{com} \quad (5.1)$$

where  $NOC(R_i)$  denotes the number of covering centers for subregion  $R_i$  and can be calculated by the tessellation algorithm. Without applying the tessellation algorithm, we can still find an upper bound on  $NOC(R_i)$  and consequently on  $L_{scan-path}(v_i)$  using lattice theory (see Appendix 5.A on how to find the upper bound). So, we have:

$$L_{scan-path}(v_i) < c_1 + \frac{c_2}{n} \quad (5.2)$$

where  $c_1$  and  $c_2$  are two constants defined in terms of  $r_{map}$  and  $r_{com}$  and whose values are defined by Eqs. 5.16 and 5.17 in Appendix 5.A. Thus, the scan path that we have found also satisfies the *proportional scan path length* property. As described in Section 5.4.2, a direct consequence of this property is that as  $n$  grows,  $\Delta_{scan}$  of each virtual node decreases. In fact,  $\Delta_{scan}$  of a virtual node  $v_i$  can be calculated as below:

$$\Delta_{scan}(v_i) = \frac{L_{scan-path}(v_i)}{v_{avg}} \quad (5.3)$$

Considering Eqs. 5.3 and 5.2 above, we find:

$$\Delta_{scan}(v_i) < \frac{1}{v_{avg}} \times \left( c_1 + \frac{c_2}{n} \right) \quad (5.4)$$

As we discuss in detail in Section 5.4.6, Eq. 5.4 plus the correctness conditions of our neighbor detector algorithm imply that as  $n$  (i.e., the number of virtual

nodes) grows, our neighbor detector algorithm remains correct with smaller values of  $\Delta_{predict}$ . Moreover, in Section 5.5.1, by using Eq. 5.4 we show that the communication cost of our neighbor detector algorithm scales linearly with the number of virtual mobile nodes.

#### 5.4.4 Neighbor Detector Algorithm

The algorithm includes two parts: a part that is executed on each real node  $p_i$  (Algorithm 5.1) and a part that is executed on each virtual node  $v_i$  (Algorithm 5.2). The algorithm relies on the movement of the virtual nodes. Thus, it divides time into rounds of duration  $\Delta_{scan}$ , where  $\Delta_{scan}$  is calculated by Eq. 5.5 and is globally known.

$$\Delta_{scan} = \Delta_{scan}(v_i) \quad \text{where } v_i \in V \quad (5.5)$$

In Eq. 5.5 above, the value of  $\Delta_{scan}(v_i)$  can be found by Eq. 5.3 of Section 5.4.3. Since the value of  $\Delta_{scan}(v_i)$  is the same for all virtual nodes, in Eq. 5.5 there is no difference which virtual node  $v_i$  is used for calculation of  $\Delta_{scan}$ .

There exist two types of rounds: collect and distribute rounds, which alternate. The first round is a collect round. Given this fact and since the execution of the algorithm starts at  $t = 0$  (i.e., when the virtual nodes start their movement by a collect scan), the collect and distribute rounds coincide with the collect and distribute scans of virtual nodes, respectively.

The algorithm proceeds in *phases*. Each phase comprises a collect and a distribute round. In the collect round, every virtual node scans its subregion and collects the location predictions sent to it by real nodes. Then, the virtual nodes share their collected location predictions with each other when the collect round terminates (i.e., when they meet at  $l_{map-center}$ ). In the distribute round, each virtual node distributes the collected location predictions to real nodes in its subregion. Every real node stores the collected location predictions that it receives to use them for neighbor detection. In the following, we discuss the algorithm in more detail and whenever necessary, we refer to the lines in the algorithm.

The algorithm keeps informed each real and virtual node of round changes via two functions GETROUND and ROUNDISOVER. Function GETROUND is called at



---

**Algorithm 5.1** Neighbor Detector Algorithm at Real Node  $p_i$ 

---

```
1: initialisation:
2:    $round \leftarrow \text{GETROUND}(\text{GETCURRENTTIME})$  { assigns to round its value at current time. The first round is a collect round }
3:    $noMsgSentInThisRound \leftarrow true$ 
4:    $networkLocs \leftarrow \perp$  { creates hash map networkLocs to store the location predictions of real nodes in the network }

5:  $\text{PRESENT}(t)$ 
6:    $N \leftarrow \emptyset$  { creates set N to store the neighbors of  $p_i$  at time  $t$  }
7:   if  $networkLocs(p_i, t) \neq \perp$  then { checks whether a location prediction for  $p_i$  at time  $t$  exists in networkLocs }
8:     for all  $p_j \in networkLocs$  do
9:       if  $p_j \neq p_i \wedge \text{DISTANCE}(networkLocs(p_j, t), networkLocs(p_i, t)) \leq r_d$  then
10:         $N \leftarrow N \cup p_j$ 
11:   return  $N$ 

12: upon  $\text{DISTANCE}(\text{GETCURRENTLOCATION}, loc(v_i, \text{GETCURRENTTIME})) \leq r_{com}$  such that  $v_i \in V$  do
13:   if  $round = collect \wedge noMsgSentInThisRound$  then
14:      $realmsg \leftarrow \perp$  { creates realmsg to encapsulate the hash map locs }
15:      $realmsg.locs \leftarrow \text{PREDICTLOCATIONS}$  { hash map locs stores the output of the mobility predictor service }
16:     trigger  $\text{BROADCAST}(realmsg, r_{com})$ 
17:      $noMsgSentInThisRound \leftarrow false$ 

18: upon  $\text{ROUNDISOVER}(\text{GETCURRENTTIME})$  do
19:    $noMsgSentInThisRound \leftarrow true$ 
20:   if  $round = collect$  then
21:      $round \leftarrow distribute$ 
22:   else if  $round = distribute$  then
23:      $round \leftarrow collect$ 

24: upon  $\text{RECEIVE}(virtmsg, v_i)$  do { receives virtmsg from virtual mobile node  $v_i$  }
25:   for all  $(p_k, t) \in virtmsg.collectedLocs$  do
26:     if  $networkLocs(p_k, t) = \perp$  then { checks if a location prediction for  $p_k$  at time  $t$  does not exist in networkLocs }
27:        $networkLocs(p_k, t) \leftarrow virtmsg.collectedLocs(p_k, t)$  { adds the location prediction for  $p_k$  at time  $t$  from collectedLocs to networkLocs }
```

---

---

**Algorithm 5.2** Neighbor Detector Algorithm at Virtual Mobile Node  $v_i$ 

---

```
28: initialisation:
29:    $round \leftarrow \text{GETROUND}(\text{GETCURRENTTIME})$  { assigns to round its value at current time. The first round is a collect round }
30:    $coveringCenters \leftarrow \{l_1, \dots, l_{NOC(R_i)}\}$  { Set coveringCenters contains the covering centers of subregion  $R_i$  }
31:    $collectedLocs \leftarrow \perp$  { creates hash map collectedLocs to store the collected location predictions }

32: upon  $\text{RECEIVE}(realmsg, p_i)$  do { receives realmsg from real node  $p_i$  }
33:   for all  $t \in realmsg.locs$  do
34:      $collectedLocs(p_i, t) \leftarrow realmsg.locs(t)$  { adds the location prediction for  $p_i$  at time  $t$  from locs to collectedLocs }

35: upon  $\text{ROUNDISOVER}(\text{GETCURRENTTIME})$  do
36:   if  $round = collect$  then
37:      $intervirtmsg \leftarrow \perp$  { creates intervirtmsg to encapsulate the hash map collectedLocs }
38:      $intervirtmsg.collectedLocs \leftarrow collectedLocs$ 
39:     trigger  $\text{BROADCAST}(intervirtmsg, r_{com})$ 
40:      $round \leftarrow distribute$ 
41:   if  $round = distribute$  then
42:      $collectedLocs.CLEAR()$  { clears the content of hash map collectedLocs at the end of each distribute round }
43:      $round \leftarrow collect$ 

44: upon  $\text{RECEIVE}(intervirtmsg, v_j)$  do { receives intervirtmsg from virtual mobile node  $v_j$  }
45:    $collectedLocs.COMBINE(intervirtmsg.collectedLocs)$  { combines  $v_i$ 's collectedLocs with collectedLocs of intervirtmsg }

46: upon  $\text{GETCURRENTLOCATION} = l_i$  such that  $l_i \in coveringCenters$  do {  $v_i$  is at a covering center of subregion  $R_i$  }
47:   if  $round = distribute$  then
48:      $virtmsg \leftarrow \perp$  { creates virtmsg to encapsulate the hash map collectedLocs }
49:      $virtmsg.collectedLocs \leftarrow collectedLocs$ 
50:     trigger  $\text{BROADCAST}(virtmsg, r_{com})$ 
```

---

initialization (lines 2 and 29). It returns the round type (collect or distribute) at current time. This value is stored in variable *round*. As stated previously, the first round is a collect round, thereby, at  $t = 0$ , GETROUND returns collect.<sup>4</sup> Function ROUNDISOVER takes current time as parameter and returns a boolean. Thus, when a round terminates ROUNDISOVER returns true, so that the value of variable *round* is changed from collect to distribute and vice-versa.

Since the trajectory function of all virtual nodes are globally known, each real node  $p_i$  can calculate its distance to every virtual node at any time. Thus, at each collect round  $p_i$  waits until its distance to a virtual node  $v_i$  becomes less than or equal to  $r_{com}$  (note that  $v_i$  can be any virtual node in  $V$ ) (line 12). Then, if  $p_i$  has not already sent a message to any virtual node in that round, it creates a message *realmmsg* to send to  $v_i$  (line 14). This message encapsulates a hash map *locs* which is used to store the output of PREDICTLOCATIONS primitive of the mobility predictor service (line 15). To store each location prediction of  $p_i$ , the hash map *locs* uses one key which is the time instant for which the location is predicted. For instance, *locs*( $t$ ) returns the predicted location at time  $t$ . Once *locs* is assigned its value, *realmmsg* is broadcast within the radius  $r_{com}$ , so it can be received by  $v_i$  (line 16).

Each virtual node has a hash map *collectedLocs*. It is used to store the location predictions that the virtual node collects. When  $v_i$  receives *realmmsg* from  $p_i$ , it stores every location prediction that exists in *locs* in its *collectedLocs* map (lines 32-34). For this storage, two keys are used where one key is the name of the real node for which the prediction is made and the other key is the time instant for which the prediction is made. For instance, *collectedLocs*( $p_i, t$ ) returns the predicted location of  $p_i$  at time  $t$ .

When a collect round terminates (i.e., when all virtual nodes are at  $l_{map-center}$ ),  $v_i$  creates *intervirtmsg* to share its *collectedLocs* with other virtual nodes (lines 35-38). It broadcasts *intervirtmsg* within the radius  $r_{com}$ , so it can be received by all virtual nodes (line 39). When a virtual node receives *intervirtmsg*, it combines its own *collectedLocs* with *collectedLocs* of *intervirtmsg*, so that at the next distribute round, all virtual nodes have the same location predictions in their *collectedLocs* maps (lines 44-45).

---

<sup>4</sup> Note that the failure of a real or virtual node is of a *crash-reboot* type i.e., if it crashes it recovers to its initial state. Therefore, calling GETROUND at initialization, enables a real or virtual node to know the round type not only at  $t = 0$  but also after each recovery.

In a distribute round,  $v_i$  encapsulates its *collectedLocs* in a *virtmsg* and broadcasts it whenever it is on a covering center of its subregion  $R_i$  (lines 46-50). The set of covering centers denoted by *coveringCenters* is defined at the initialization (line 30) and can be found by the tessellation algorithm discussed in Section 5.4.3. Thus,  $v_i$  broadcasts *virtmsg* on covering centers so that they are disseminated in the whole  $R_i$ .

Each real node has a hash map called *networkLocs* that is used to store the location predictions of all real nodes in the network. Similarly to *collectedLocs*, *networkLocs* has two keys to store a location prediction: one key is the name of the real node for which the prediction is made and the other key is the time instant for which the prediction is made. For instance, *networkLocs*( $p_i, t$ ) returns the predicted location of  $p_i$  at time  $t$ . The *networkLocs* map is augmented in distribute rounds, i.e., when new location predictions are received in *collectedLocs* of a *virtmsg* (lines 24-27). Thus, whenever primitive `PRESENT( $t$ )` is invoked at  $p_i$ , the map lookups on *networkLocs* as well as distance comparisons are performed to find the real nodes which are in the neighborhood region of  $p_i$  at time  $t$  (lines 5-9). The names of real nodes found in this way, are stored in set  $N$  which is returned as the result (lines 10-11).

### 5.4.5 Proof of Correctness

In this section we present a proof of correctness for the algorithm. As we show, under certain conditions, the algorithm correctly implements the time-limited neighbor detector abstraction (defined in Section 5.3) and can tolerate the failure of one to all virtual mobile nodes. Thus, we start by describing the intuition behind the proof and some of the conditions required to guarantee the correctness of the algorithm. We also introduce some preliminary notations, definitions and lemmas that are used throughout the proof. Then, we present the proof. In particular, we define the minimum  $\Delta_{predict}$  for which the algorithm is correct. This value is then used in Section 5.4.6, where we study the impact of increasing the number of virtual mobile nodes on it.

#### 5.4.5.1 Intuition behind the Proof

As we show in the proof, the algorithm can tolerate the failure of one to all virtual nodes under certain conditions. Recall that the failure of a virtual node is of a *crash-reboot* type: it crashes when the area around its trajectory becomes unpopulated and it recovers to its initial state as soon as it reenters a dense area. Since the algorithm proceeds in phases, to guarantee its correctness in spite of the failure of the virtual nodes, our intuition is as follows. We prove the correctness of the algorithm for a category of the executions called *nice executions*, which basically correspond to the executions in periodically well-populated regions such as main squares in a downtown area. In a nice execution, virtual nodes can fail. However, there exist time periods during a nice execution where the whole region  $R$  is populated well enough so that all virtual nodes are up. Each of such periods is long enough to contain at least one phase that is entirely executed in it.<sup>5</sup> A phase that is executed while all virtual nodes are up is called an *atomic phase*. In an atomic phase, no virtual node crashes, thereby, no location prediction is lost by a virtual node during the collection, sharing and distribution of location predictions. Thus, for neighbor detection, a real node  $p_i$  should rely on the location predictions that it receives during the distribute round of each atomic phase. Intuitively, this means that the location predictions that  $p_i$  receives during the distribution in an atomic phase should be long enough so that  $p_i$  can use them for current and future neighbor detection at least until the distribution in the next atomic phase (as for the past neighbor detection,  $p_i$  can use the location predictions that it has received and stored in all previous atomic phases). Note that, in the distribute round of an atomic phase,  $p_i$  may receive the location predictions at the latest at the end of the round. Therefore, to guarantee the correctness of the algorithm,  $\Delta_{predict}$  should be long enough to ensure the current and future neighbor detection by  $p_i$ , at least, at each time instant between the end of the distribute rounds of two consecutive atomic phases. As we further show, such  $\Delta_{predict}$  can be

---

<sup>5</sup> The existence of nice executions is realistic considering the variation of population density in a periodically well-populated urban region (e.g., a public square) during a time interval (e.g., a working day). In fact, in a periodically well-populated urban region, there are periods of time where the population density becomes so low so that the virtual nodes that scan the region become unstable (i.e., they crash and recover many times while traveling through their preplanned trajectory). However, after some bounded time duration, the population density increases high enough to guarantee that the virtual nodes remain up for at least some period of time.

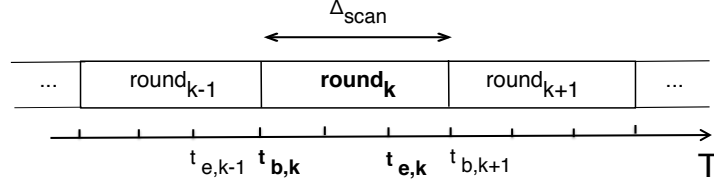


Fig. 5.3:  $t_{b,k}$  and  $t_{e,k}$  of  $round_k$

found based on the maximum time duration between the end of the distribute round of two consecutive atomic phases.

#### 5.4.5.2 Preliminaries

Before beginning the main part of the proof, we present some preliminary notations, definitions and lemmas that are used throughout the proof. In particular, we focus on formally defining a nice execution and highlight those characteristics of a nice execution that can be used for the proof.

In the following, we first present the preliminary notations and definitions. We then present the preliminary lemmas.

**Preliminary Notations and Definitions.** Here we present the preliminary notations and definitions.

- Given two sets  $A$  and  $B$ ,  $A \subseteq B$  indicates that  $A$  is a subset of  $B$ .
- $P_R$  denotes a subset of  $P$  (recall that  $P$  is the set of all real nodes) such that  $\forall p_i \in P_R$ ,  $p_i$  never leaves region  $R$  and the movement of  $p_i$  during  $\Delta_{scan}$  is negligible.
- $round_k$  denotes the  $k$ th round of the algorithm, where  $k$  (also called *the index of the round*) is an integer such that  $k \geq 1$ .
- $\phi_i$  denotes the  $i$ th phase of the algorithm, where  $i$  is an integer such that  $i \geq 1$ .
- $d(\phi_i)$  returns the index of the distribute round of a phase  $\phi_i$ . For instance, if  $round_k$  is the distribute round of  $\phi_i$ , then  $d(\phi_i) = k$ .
- $t_{b,k}$  and  $t_{e,k}$  refer to the first clock tick and the last clock tick in  $round_k$ , respectively (See Fig. 5.3). Note that,  $t_{e,k} = t_{b,k} + \Delta_{scan} - 1$ . We call  $t_{b,k}$ , *the beginning time* of  $round_k$  and  $t_{e,k}$  *the end time* of  $round_k$ .
- *Global System State.* The local state of a (virtual or real) node is a tuple that contains the value of its variables. In particular, among these variables there is a

variable which indicates whether the node is up or down. The global system state is a vector  $\sigma$  whose elements are the local states of all virtual and real nodes in the system.

- *Execution.* An execution  $E$  of neighbor detector algorithm is an infinite sequence that maps every time instant in  $T$  to a global system state. Formally,  $E := (\sigma_t)_{t=0}^{+\infty}$  where  $\sigma_t$  is the global state at time  $t \in T$ .
- *Stable and unstable periods.* Let  $E$  be an execution, then  $E$  could include two types of time periods called *stable* and *unstable* periods. A stable period is a period during which all virtual nodes are up. On the other hand, an unstable period is a period during which at least one virtual node is down.
- *Nice Execution.* Let  $E$  be an execution. Let  $\Delta_{stable}^{min}$  and  $\Delta_{unstable}^{max}$  be two non-negative integers such that  $\Delta_{stable}^{min} = 4 \times \Delta_{scan}$ . We say that  $E$  is *nice* if the duration of each stable period in  $E$  is at least equal to  $\Delta_{stable}^{min}$  and the duration of each unstable period in  $E$  is at most equal to  $\Delta_{unstable}^{max}$ . Moreover, the first stable period in  $E$  starts at  $t = 0$ .
- *Atomic phase.* Let  $E$  be an execution. Let  $\phi_i$  be a phase in  $E$ . Then  $\phi_i$  is *atomic* if it entirely occurs in a stable period of  $E$ , that is, while all virtual nodes are up.
- *Nonatomic phase.* Let  $E$  be an execution. Let  $\phi_i$  be a phase in  $E$ . Then  $\phi_i$  is *nonatomic* if at least a part of it occurs in an unstable period of  $E$ .

**Preliminary Lemmas.** We prove seven preliminary lemmas where two lemmas, i.e., Lemma 5.3 and Lemma 5.6, have each an associated corollary. Lemmas 5.1 to 5.5 are straightforward and mainly used to prove (directly or indirectly) Lemmas 5.6 and 5.7. Lemmas 5.6 and 5.7 are important results which are used (with Lemma 5.3 and its associated corollary) for the main proof in the next section. In particular, Lemma 5.6 proves that there exists a maximum, denoted by  $\Delta_{gap}$ , for the time duration between the end of the distribute rounds of two consecutive atomic phases in a nice execution. It also defines the value of  $\Delta_{gap}$ . Lemma 5.7 shows that in a nice execution the time duration between the end of a round  $round_k$  such that  $k \geq 3$  and the end of the distribute round of the last atomic phase before  $round_k$  has a maximum which is equal to  $\Delta_{gap}$ .

In the following, we prove the lemmas and whenever necessary, we give some additional information regarding their use and the intuition behind them.

**Lemma 5.1.** *Let  $E$  be an execution. If  $E$  is nice, then every stable period in  $E$  contains at least one atomic phase.*

*Proof.* If  $E$  is nice, then the duration of every stable period in  $E$  is greater than or equal to  $4 \times \Delta_{scan}$ . Therefore, regardless of how the rounds occur in a stable period, the stable period contains at least one phase which is entirely executed in it. Hence, in this case each stable period contains at least one atomic phase.  $\square$

**Lemma 5.2.** *Let  $E$  be an execution. If all virtual nodes are correct, then  $E$  is nice.*

*Proof.* If all virtual nodes are correct, then they are always up. This means that there exists only one stable period in  $E$  which has an infinite duration. Therefore, the duration of the stable period is greater than  $\Delta_{stable}^{min} = 4 \times \Delta_{scan}$  and the duration of any unstable period is zero and consequently less than or equal to  $\Delta_{unstable}^{max}$ .  $\square$

**Lemma 5.3.** *Let  $E$  be an execution. If  $E$  is nice, then the first phase or  $\phi_1$  of the algorithm is also the first atomic phase in  $E$ .*

*Proof.* The first phase of the algorithm denoted by  $\phi_1$  comprises  $round_1$  and  $round_2$ . By definition, the first stable period of a nice execution begins at  $t = 0$ . Moreover, every stable period of a nice execution lasts at least  $4 \times \Delta_{scan}$ . Therefore, if  $E$  is nice, then the first four rounds of  $E$  are in the first stable period. Accordingly, phase  $\phi_1$  occurs entirely in the first stable period and thus, it is atomic. Therefore,  $\phi_1$  is also the first atomic phase.  $\square$

**Corollary 5.1.** *Let  $E$  be an execution and  $round_k$  be a round in  $E$  such that  $k \geq 3$ . If  $E$  is nice, then there exists at least one atomic phase in  $E$  which occurs before  $round_k$ .*

*Proof.* The proof follows directly from Lemma 5.3.  $\square$

**Lemma 5.4.** *Let  $E$  be an execution and  $S_i$  be a stable period in  $E$ . If there exists more than one atomic phase in  $S_i$ , then there exists no nonatomic phase in  $S_i$  that occurs between the atomic phases.*

*Proof.* Let  $\phi_i$  and  $\phi_j$  be two atomic phases in  $S_i$  such that  $\phi_j$  is the next atomic phase after  $\phi_i$ . Assume for contradiction that there exists a nonatomic phase  $\phi_i + 1$  after  $\phi_i$  and before  $\phi_j$  in  $S_i$ . Since  $\phi_i + 1$  is nonatomic, a part of it should occur in an unstable period. This suggests that an unstable period should exist between  $\phi_i$  and  $\phi_j$ , which implies that  $\phi_j$  should occur in a stable period different than  $S_i$  which is impossible.  $\square$

**Lemma 5.5.** *Let  $E$  be a nice execution. Then, every round in  $E$  is either in an atomic phase or between two atomic phases.*

*Proof.* Let  $round_k$  be a round in  $E$ . From Lemma 5.3, we get that the first phase in  $E$  is atomic, which means that  $round_1$  and  $round_2$  are in an atomic phase. Thus, the lemma holds for  $k < 3$ . To show that the lemma also holds for  $k \geq 3$  we proceed as follows. By Corollary 5.1 we know that if  $k \geq 3$ , there exists at least one atomic phase before  $round_k$ . Thus, to prove that the lemma holds for  $k \geq 3$ , we should show that  $round_k$  is either in an atomic phase or there exists an atomic phase after  $round_k$  so that based on Corollary 5.1, we can conclude that  $round_k$  is between two atomic phases. In the following, we prove the lemma for  $k \geq 3$  by considering two possible cases.

**Case 1:  $round_k$  is in an unstable period of  $E$ .** Since an unstable period of  $E$  lasts at most  $\Delta_{unstable}^{max}$ , we know that there exists a stable period after the unstable period, which according to Lemma 5.1 contains at least one atomic phase. Therefore, in this case there exists an atomic phase after  $round_k$  and the lemma holds.

**Case 2:  $round_k$  is in a stable period of  $E$ .** Let  $S_i$  denote the stable period where  $round_k$  occurs. By Lemma 5.1, we know that  $S_i$  contains at least one atomic phase. So there exist two subcases: (1)  $round_k$  is in an atomic phase of  $S_i$  and the lemma holds; (2)  $round_k$  is outside of any atomic phases of  $S_i$ , which means that  $round_k$  is in a nonatomic phase that partly occurs in  $S_i$ . By Lemma 5.4, we know that the nonatomic phase that contains  $round_k$  cannot occur between two atomic phases of  $S_i$ . Therefore,  $round_k$  is either outside and before any atomic phases of  $S_i$  or outside and after any atomic phases of  $S_i$ . We show that in both cases the lemma holds. In fact, if  $round_k$  is outside and before any atomic phases of  $S_i$ , then it means that there exists an atomic phase after  $round_k$  and the lemma holds. If  $round_k$  is outside and after any atomic phases of  $S_i$ , then it means that there exists an unstable period just after  $S_i$  (otherwise,  $round_k$  should be in an atomic phase). Since an unstable period of  $E$  lasts at most  $\Delta_{unstable}^{max}$ , we know that there exists a stable period after the unstable period, which according to Lemma 5.1 contains at least one atomic phase. Therefore, there exists an atomic phase after  $round_k$  and the lemma holds.  $\square$

Now that we have proved Lemmas 5.1 to 5.5, we can present more important results based on them. The next lemma proves that there exists a maximum, denoted



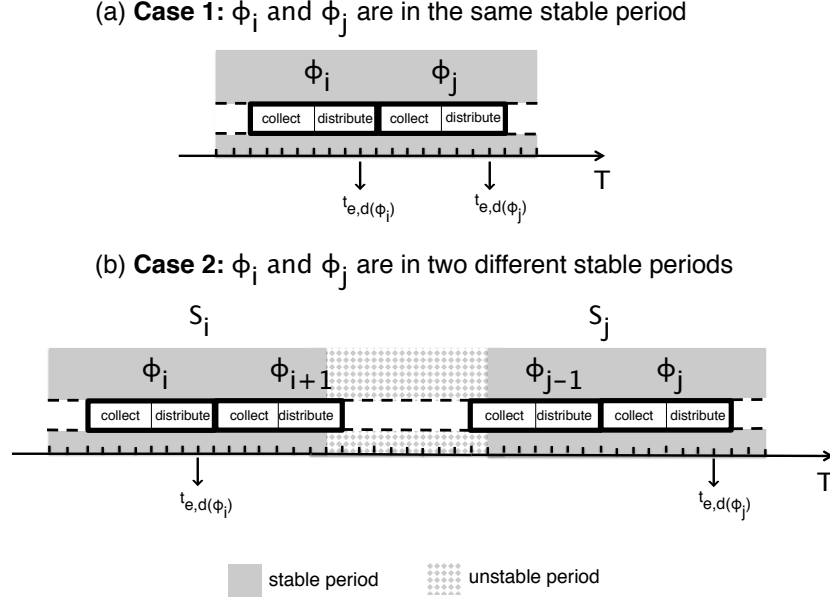


Fig. 5.4: Examples of occurrence of two consecutive atomic phases  $\phi_i$  and  $\phi_j$  in a nice execution. The cases correspond to the cases of the proof of Lemma 5.6.

by  $\Delta_{gap}$ , for the time duration between the end of the distribute rounds of two consecutive atomic phases in a nice execution. It also defines the value of  $\Delta_{gap}$ .

**Lemma 5.6.** *Let  $E$  be an execution. Let  $\phi_i$  and  $\phi_j$  be two atomic phases in  $E$  such that  $\phi_j$  is the next atomic phase after  $\phi_i$ . If  $E$  is nice, then the time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,d(\phi_j)}$  has a maximum denoted by  $\Delta_{gap}$  such that  $\Delta_{gap} = 6 \times \Delta_{scan} - 2 + \Delta_{unstable}^{max}$ .*

*Proof.* We assume that  $E$  is nice and we consider the two possible cases below.

**Case 1:  $\phi_i$  and  $\phi_j$  are in the same stable period.** Since  $\phi_j$  is the next atomic phase after  $\phi_i$  and since  $\phi_i$  and  $\phi_j$  are in the same stable period, by Lemma 5.4, we know that there exists no nonatomic phase between  $\phi_i$  and  $\phi_j$ . Therefore,  $\phi_j$  can only be the phase just after  $\phi_i$  (see Fig. 5.4.a). Thus, in this case the time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,d(\phi_j)}$  is always equal to  $2 \times \Delta_{scan}$  and the maximum time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,d(\phi_j)}$  is also equal to  $2 \times \Delta_{scan}$ .

**Case 2:  $\phi_i$  and  $\phi_j$  are in two different stable periods.** Let  $S_i$  and  $S_j$  be two different stable periods such that  $\phi_i$  is in  $S_i$  and  $\phi_j$  is in  $S_j$ . Since  $\phi_j$  is the next atomic phase after  $\phi_i$ , we know that there is no atomic phase between  $\phi_i$  and  $\phi_j$ .

Moreover, since  $E$  is nice by Lemma 5.1 we know that every stable period in  $E$  contains at least one atomic phase. Therefore, there exists no stable period between  $S_i$  and  $S_j$ . Accordingly, there exists only one unstable period between  $S_i$  and  $S_j$ . Thus, in this case the maximum time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,d(\phi_j)}$  corresponds to the following situation: the unstable period between  $S_i$  and  $S_j$  lasts  $\Delta_{unstable}^{max}$ . In addition, there exist a nonatomic phase  $\phi_{i+1}$  just after  $\phi_i$  and a nonatomic phase  $\phi_{j-1}$  just before  $\phi_j$  such that one time unit of  $\phi_{i+1}$  and one time unit of  $\phi_{j-1}$  occur in the unstable period between  $S_i$  and  $S_j$  and  $2 \times \Delta_{scan} - 1$  time units of  $\phi_{i+1}$  occur in  $S_i$  and  $2 \times \Delta_{scan} - 1$  time units of  $\phi_{j-1}$  occur in  $S_j$  (see Fig. 5.4.b).<sup>6</sup> Thus, the maximum time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,d(\phi_j)}$  in this case is equal to  $2 \times (2 \times \Delta_{scan} - 1) + \Delta_{unstable}^{max} + 2 \times \Delta_{scan} = 6 \times \Delta_{scan} - 2 + \Delta_{unstable}^{max}$ .

In each case described above, the time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,d(\phi_j)}$  has a maximum. In Case 1, the maximum is equal to  $2 \times \Delta_{scan}$ . In Case 2, it is equal to  $6 \times \Delta_{scan} - 2 + \Delta_{unstable}^{max}$ . Hence,  $\Delta_{gap}$  is equal to  $6 \times \Delta_{scan} - 2 + \Delta_{unstable}^{max}$ .  $\square$

The value of  $\Delta_{gap}$  defined by Lemma 5.6 corresponds to nice executions in general. The following corollary of Lemma 5.6 defines the value of  $\Delta_{gap}$  in a special case of nice executions, i.e., where all virtual nodes are correct. This value of  $\Delta_{gap}$  is smaller than the value defined in Lemma 5.6. We will use this value in Section 5.4.5.3 to define the minimum  $\Delta_{predict}$  required for the correctness of the algorithm in the special case where all virtual nodes are correct.

**Corollary 5.2.** *If all virtual nodes are correct, then  $\Delta_{gap} = 2 \times \Delta_{scan}$ .*

*Proof.* Let  $E$  be the execution considered in Lemma 5.6. Let  $\phi_i$  and  $\phi_j$  be the two atomic phases considered in Lemma 5.6. From Lemma 5.2, we know that if all virtual nodes are correct, then  $E$  is nice. Moreover, if all virtual nodes are correct,  $E$  contains only one stable period and no unstable period. Therefore, there exists only one case, i.e.,  $\phi_i$  and  $\phi_j$  are in the same stable period. By proof of Lemma 5.6, we know that the maximum time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,d(\phi_j)}$  in this case is equal to  $2 \times \Delta_{scan}$ . Hence,  $\Delta_{gap}$  is equal to  $2 \times \Delta_{scan}$ .  $\square$

Our final preliminary lemma shows that in a nice execution the time duration between the end of a round  $round_k$  such that  $k \geq 3$  and the end of the distribute round of the last atomic phase before  $round_k$  has a maximum. This maximum is equal to the time duration  $\Delta_{gap}$  which is already defined by Lemma 5.6. Note that

<sup>6</sup> Recall that the duration of a phase is equal to  $2 \times \Delta_{scan}$  time units.

the following lemma is defined for  $k \geq 3$  since by Corollary 5.1 we know that there exists at least one atomic phase before  $round_k$  if  $k \geq 3$ .

**Lemma 5.7.** *Let  $E$  be a nice execution. Let  $round_k$  be a round in  $E$  such that  $k \geq 3$ . Let  $\phi_i$  be the last atomic phase before  $round_k$ . Then, the maximum time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,k}$  is equal to  $\Delta_{gap}$ .*

*Proof.* By Lemma 5.5,  $round_k$  occurs either in an atomic phase or between two atomic phases. Thus, let  $\phi_j$  be the atomic phase just after  $\phi_i$ , we know that  $round_k$  occurs either between  $\phi_i$  and  $\phi_j$  or in  $\phi_j$ . Therefore, we can say that  $round_k$  can be, at the latest, the distribute round of  $\phi_j$ . From Lemma 5.6, we get that the maximum time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,d(\phi_j)}$  is equal to  $\Delta_{gap}$ . Hence, the lemma holds.  $\square$

### 5.4.5.3 The Proof

We prove the correctness of the algorithm under a set of conditions. In particular, we define the minimum  $\Delta_{predict}$  for which the algorithm is correct in different cases of nice executions i.e., in the general case as well as in the special case where all virtual mobile nodes are correct. Thus, in the following, we first introduce the conditions under which the algorithm is correct and formally describe the meaning and the implication of each condition. We then introduce the theorems and the lemmas that are used for the proof.

Note that in this section,  $\Delta_{predict}$  refers to the prediction interval of the mobility predictor service (defined in Section 5.2.4),  $\Delta_{future}$  refers to the time duration defined in the *time-limited completeness* property of the neighbor detector (stated in Section 5.3.1.2) and  $\Delta_{gap}$  refers to the time duration defined in Lemma 5.6.

**Conditions.** We prove that the algorithm is correct under the conditions listed hereafter.

- C1 We consider a nice execution for the proof.
- C2 The value of the constant  $d$  defined in the *reliable delivery* property of the LocalCast service (stated in Section 5.2.2) is negligible.
- C3 The execution time of lines 36-40 and line 45 of the algorithm is negligible.
- C4  $\Delta_{predict} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$ .

- C5 Let  $p_i$  and  $p_j$  be the processes defined in the *time-limited completeness* property of the neighbor detector (stated in Section 5.3.1.2), then  $p_i, p_j \in P_R$ .
- C6 If PRESENT( $t$ ) of the neighbor detector is called, then  $t \geq t_{e,1}$  and  $t_c \geq t_{b,3}$  where  $t_c$  is the time when PRESENT( $t$ ) is called.

Informally speaking, Condition C1 enables us to use the characteristics of a nice execution (stated in the preliminary lemmas of Section 5.4.5.2) to prove the correctness of the algorithm. Condition C2 plus the *reliable delivery* property of the LocalCast service (stated in Section 5.2.2) ensure that a node which remains for a negligible time within the broadcast radius of the sender, will deliver the broadcast message with negligible delay. Condition C3 guarantees that the creation of a *intervirtmsg* message (i.e., the message used by a virtual node to share its collected location predictions with other virtual nodes) and the combination of the location predictions collected by different virtual nodes takes a negligible time. This condition plus some other properties imply that the sharing of collected location predictions between virtual nodes takes a negligible time. Condition C4 defines a value of  $\Delta_{predict}$  for which the algorithm is correct. This value is long enough to ensure the current and future neighbor detection by a real node at each time instant between the end of the distribute rounds of two consecutive atomic phases. As we show in the proof, the value defined in Condition C4 is in fact the minimum value of  $\Delta_{predict}$  for which the algorithm is correct. Condition C5, assumes that processes  $p_i$  and  $p_j$  defined in the *time-limited completeness* property of the neighbor detector are in set  $P_R$ . This means that  $p_i$  and  $p_j$  are always in region  $R$  and their movements are negligible during  $\Delta_{scan}$ . As we show in the poof, this condition plus some other properties ensure that in the collect as well as in the distribute round of an atomic phase, there exists a time when  $p_i$  and  $p_j$  are within distance  $r_{com}$  to a virtual node and thus, can communicate with it. Finally, Condition C6 implies that the neighbor detection is not guaranteed for time instants before  $t_{e,1}$  (i.e., the end time of  $round_1$ ) and PRESENT( $t$ ) is called in  $round_k$  where  $k \geq 3$ .

**Theorems and Lemmas.** We prove four theorems. The main theorem, which proves the correctness of the algorithm, is Theorem 5.3. The proof of Theorem 5.3 relies on Theorem 5.1 and Theorem 5.2. Theorem 5.1 proves that the algorithm satisfies the *time-limited completeness* property of the time-limited neighbor detector abstraction (stated in Section 5.3.1.2). To prove Theorem 5.1, we use three helper Lemmas, that is, Lemma 5.8, Lemma 5.9 and Lemma 5.10. Each helper lemma also

relies for its proof on some of the preliminary lemmas introduced in the previous section. Theorem 5.2 proves that the algorithm satisfies the *perfect accuracy* property of the time-limited neighbor detector abstraction (stated in Section 5.3.1.2). The proof of this theorem is straightforward and relies on no lemma. Our final theorem, Theorem 5.4, proves that the minimum  $\Delta_{predict}$  for which the algorithm is correct is equal to the one defined in Condition C4. The proof of this theorem relies on Theorem 5.3, Theorem 5.1 and Corollary 5.4, which is the associated corollary of Lemma 5.10.

In the following, we first prove the helper lemmas for Theorem 5.1. We then prove Theorems 5.1 to 5.4, respectively. Note that throughout the proof, whenever necessary we refer to the lines of the algorithm.

Before presenting the helper lemmas for Theorem 5.1, we describe the key idea behind the proof of Theorem 5.1. As previously stated according to Theorem 5.1, the algorithm satisfies the *time-limited completeness* property. Roughly speaking, this property requires a process  $p_i$  to detect any process  $p_j$  that is in the neighborhood region of  $p_i$  at any time in the past, present and up to interval  $\Delta_{future}$  in the future. In the algorithm a process uses the location predictions that it stores in its *networkLocs* for neighbor detection. Thereby, in order to prove Theorem 5.1, by using the helper lemmas we basically prove that: (1) at the distribute round of each atomic phase,  $p_i$  receives accurate location predictions for both  $p_i$  and  $p_j$  and stores the predictions in its *networkLocs*; (2) the location predictions stored in *networkLocs* are long enough so that at any time instant in a *round<sub>k</sub>* such that  $k \geq 3$ ,  $p_i$  has enough predictions to detect past, present and future neighbors. The reason why  $k \geq 3$ , is that in a nice execution, the first distribute round which occurs in an atomic phase is *round<sub>2</sub>*. The helper lemmas 5.9 and 5.10 each have a corollary. These corollaries are later used by Theorem 5.4 to prove that the value of  $\Delta_{predict}$  defined in Condition C4, is also the minimum  $\Delta_{predict}$  for which the algorithm is correct.

**Lemma 5.8.** *Let  $p_i$  and  $p_j$  be the processes defined in the time-limited completeness property of the neighbor detector. Let *round<sub>k</sub>* be the distribute round of an atomic phase. Then, in *round<sub>k</sub>*, process  $p_i$  receives a *virtmsg* from a virtual mobile node with *collectedLocs* map which contains accurate location predictions for both  $p_i$  and  $p_j$ . Moreover, all location predictions are defined for the time interval  $[t_{e,k-1}; t_{b,k-1} + \Delta_{predict}]$ .*

*Proof.* If *round<sub>k</sub>* is a distribute round in an atomic phase, then *round<sub>k-1</sub>* is a collect round in the same atomic phase. According to Condition C5,  $p_i, p_j \in P_R$ . Therefore,

$p_i$  and  $p_j$  are always in region  $R$  and their movements are negligible during  $\Delta_{scan}$ . Moreover, the scan path of each virtual node guarantees the *scan completeness* property (stated in Section 5.4.2) which implies that in each round, for each location  $l$  in a subregion  $R_i$  scanned by virtual node  $v_i$ , there exists a time when  $l$  is within distance  $r_{com}$  to  $v_i$ . Considering these facts and since in  $round_{k-1}$  all virtual nodes are up, then in  $round_{k-1}$  there exists a time when the distance of  $p_i$  and  $p_j$  to a virtual node becomes less than or equal to  $r_{com}$ . According to the algorithm, in a collect round, as soon as a real node realizes that is within a distance  $r_{com}$  to a virtual node, it sends its location predictions in *locs* map to the virtual node (lines 12-16). Therefore, in  $round_{k-1}$  both  $p_i$  and  $p_j$  send their *locs* maps to a virtual node. The *strong accuracy* property of the mobility predictor service (stated in Section 5.2.4) guarantees that the location predictions of  $p_i$  and  $p_j$  in their *locs* maps are accurate. Moreover, in  $round_{k-1}$ , if a real node is within a distance  $r_{com}$  to a virtual node at the earliest possible time (i.e., at  $t_{b,k-1}$  or the beginning time of the round), its *locs* map is defined for time interval  $T_1 = [t_{b,k-1}; t_{b,k-1} + \Delta_{predict}]$ . On the other hand, if in  $round_{k-1}$  a real node is within a distance  $r_{com}$  to a virtual node at the latest possible time (i.e.,  $t_{e,k-1}$  or the end time of the round), its *locs* map is defined for time interval  $T_2 = [t_{e,k-1}; t_{e,k-1} + \Delta_{predict}]$ . The intersection of  $T_1$  and  $T_2$  is  $T_3 = [t_{e,k-1}; t_{b,k-1} + \Delta_{predict}]$ . Thus, regardless of the time when  $p_i$  and  $p_j$  are within a distance  $r_{com}$  to a virtual node in  $round_{k-1}$ , their *locs* maps contain location predictions for time interval  $T_3$ . Condition C2 plus the *reliable delivery* property of the underlying broadcast (stated in Section 5.2.2) ensure that a node which remains for a negligible time within the broadcast radius of the sender, will deliver the broadcast message with negligible delay. Thus, considering Condition C2, the *reliable delivery* property of the underlying broadcast and the fact that in  $round_{k-1}$  all virtual nodes are up, we know that the communication between a virtual node and a correct real node in  $round_{k-1}$  is reliable and takes negligible delay. Moreover, according to the algorithm, a virtual node stores all the location predictions received from the real nodes in its *collectedLocs* map (lines 32-34). Therefore, the location predictions sent by  $p_i$  and  $p_j$  in  $round_{k-1}$  are received and stored by the virtual nodes which are in their proximity in  $round_{k-1}$ .

According to the algorithm, when  $round_{k-1}$  terminates, virtual nodes share their own *collectedLocs* with each other by broadcasting *intervirtmsg* messages (lines 36-40). Then, they combine the received *collectedLocs* with their own *collectedLocs* (line 45). Considering Condition C2, the *reliable delivery* property of the underly-

ing broadcast and the fact that all virtual nodes are up and meet when  $round_{k-1}$  (which is a collect round) terminates, we know that the communication between virtual nodes is reliable and takes negligible time. Condition C3 also guarantees that the creation of a *intervirtmsg* message and the combination of *collectedLocs* maps takes a negligible time. As a result, at the beginning of  $round_k$ , which is a *distribute* round, all the virtual nodes have the location predictions of  $p_i$  and  $p_j$  in their *collectedLocs*. According to the algorithm, in  $round_k$  each virtual node  $v_i$  encapsulates its *collectedLocs* map in a *virtmsg* and broadcasts it at the covering centers of its subregion  $R_i$  (lines 46-50). As previously stated, according to Condition C5,  $p_i \in P_R$ , which means that  $p_i$  is always in  $R$  and its movement during  $\Delta_{scan}$  is negligible. We also know that in  $round_k$ , all virtual nodes are up. Therefore, regardless of the subregion where  $p_i$  is found, there exists a time in  $round_k$  when  $p_i$  is within the broadcast radius of a virtual node which broadcasts *virtmsg*. Condition C2 and the *reliable delivery* property of the underlying broadcast, also guarantee that *virtmsg* will be received by  $p_i$  in a negligible time. Therefore, we know that regardless of the subregion where  $p_i$  is found in  $round_k$ , it receives a *virtmsg* encapsulating the *collectedLocs* and broadcast by a virtual node, in  $round_k$ . As we have just shown, the *collectedLocs* map contains accurate location predictions for both  $p_i$  and  $p_j$  and all the location predictions are defined for time interval  $T_3$ . Hence, the Lemma holds.  $\square$

**Lemma 5.9.** *Let  $p_i$  and  $p_j$  be the processes defined in the time-limited completeness property of the neighbor detector. Let phase  $\phi_i$  be an atomic phase. Then, once  $\phi_i$  terminates, *networkLocs* of  $p_i$  contains accurate location predictions for both  $p_i$  and  $p_j$ , which are all defined for the time interval  $[t_{e,1}; t_{e,d(\phi_i)} + \Delta_{gap} + \Delta_{future}]$ .*

*Proof.* For the proof we use induction.

**Base Case: The first atomic Phase.** From Condition C1, we get that the execution that we consider for the proof is nice. Thus, from Lemma 5.3, we get that in a nice execution, the first phase or  $\phi_1$  is also the first atomic phase. According to the algorithm,  $\phi_1$  comprises  $round_1$  and  $round_2$  where  $round_1$  is a collect round and  $round_2$  is a distribute round. Since  $round_2$  is a distribute round of an atomic phase, by Lemma 5.8 we know that in  $round_2$ ,  $p_i$  receives the accurate location predictions for both  $p_i$  and  $p_j$  and the predictions are all defined for time interval  $T_1 = [t_{e,1}; t_{b,1} + \Delta_{predict}]$ . By replacing  $\Delta_{predict}$  by its value defined in Condition C4, we have  $T_1 = [t_{e,1}; t_{b,1} + 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}] = [t_{e,1}; t_{e,2} + \Delta_{gap} + \Delta_{future}]$ .

Since  $t_{e,2} = t_{e,d(\phi_1)}$ , we have  $T_1 = [t_{e,1}; t_{e,d(\phi_1)} + \Delta_{gap} + \Delta_{future}]$ . According to the algorithm,  $p_i$  stores all the received location predictions in *networkLocs* and uses them for neighbor detection (lines 24-27). Therefore, once the first atomic phase terminates, process  $p_i$  has the accurate location predictions for both  $p_i$  and  $p_j$  which are all defined for the time interval  $T_1$ . Hence, the lemma holds in this case.

**Inductive Step.** We assume that the lemma holds for an atomic phase  $\phi_i$  which is either the first atomic phase or after the first atomic phase. Then, we wish to show that the lemma holds for  $\phi_j$  which is the next atomic phase after  $\phi_i$ . By inductive hypothesis we know that once  $\phi_i$  terminates, *networkLocs* of  $p_i$  contains accurate location predictions for both  $p_i$  and  $p_j$ , which are all valid for the time interval  $T_1 = [t_{e,1}; t_{e,d(\phi_i)} + \Delta_{gap} + \Delta_{future}]$ . According to Condition C1, the execution that we consider is nice, therefore, by Lemma 5.6, we know that the maximum time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,d(\phi_j)}$  is equal to  $\Delta_{gap}$ . Thus, let  $T_2 = [t_{e,1}; t_{e,d(\phi_j)} + \Delta_{future}]$ , we know that once  $\phi_i$  terminates, *networkLocs* of  $p_i$  contains accurate location predictions for both  $p_i$  and  $p_j$ , which are valid for the time interval  $T_2$  since  $T_2 \subseteq T_1$ . Moreover, by Lemma 5.8, we know that in the distribute round of  $\phi_j$ , the process  $p_i$  receives the accurate location predictions for both  $p_i$  and  $p_j$  and the predictions are defined for the time interval  $T_3 = [t_{e,d(\phi_j)-1}; t_{b,d(\phi_j)-1} + \Delta_{predict}]$ . By replacing  $\Delta_{predict}$  by its value defined in Condition C4, we have  $T_3 = [t_{e,d(\phi_j)-1}; t_{b,d(\phi_j)-1} + 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}] = [t_{e,d(\phi_j)-1}; t_{e,d(\phi_j)} + \Delta_{gap} + \Delta_{future}]$ . According to the algorithm,  $p_i$  stores all the received location predictions in *networkLocs* and uses them for neighbor detection (lines 24-27). Thus, when  $\phi_j$  terminates, *networkLocs* of  $p_i$  contains accurate location predictions for both  $p_i$  and  $p_j$ , which are defined for time interval  $T_4 = T_2 \cup T_3 = [t_{e,1}; t_{e,d(\phi_j)} + \Delta_{gap} + \Delta_{future}]$ . Hence, the lemma holds in this case.

Since the base case holds and since the inductive step holds, the lemma holds.  $\square$

**Corollary 5.3.** *Let  $\Delta_{predict}^{min}$  be the minimum value of  $\Delta_{predict}$  for which Lemma 5.9 holds, then  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$  (i.e., the value defined in Condition C4).*

*Proof.* For the proof we show that if  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$ , then Lemma 5.9 holds for  $\Delta_{predict} \geq \Delta_{predict}^{min}$  and it does not hold for  $\Delta_{predict} < \Delta_{predict}^{min}$ . Thus, we assume that  $\Delta_{predict} = \Delta_{predict}^{min} + \Delta_{diff}$  where  $\Delta_{diff} = \Delta_{predict} - \Delta_{predict}^{min}$  and we consider the two following cases:



**Case 1.**  $\Delta_{predict} \geq \Delta_{predict}^{min}$ . Consider the proof of Lemma 5.9. Then, in Base Case of the induction to calculate  $T_1$ , replace  $\Delta_{predict}$  by  $\Delta_{predict}^{min} + \Delta_{diff}$ . If  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$ , then  $T_1 = [t_{e,1}; t_{e,2} + \Delta_{gap} + \Delta_{future} + \Delta_{diff}]$ . As  $t_{e,2} = t_{e,d(\phi_1)}$ , we have  $T_1 = [t_{e,1}; t_{e,d(\phi_1)} + \Delta_{gap} + \Delta_{future} + \Delta_{diff}]$ . Since in this case  $\Delta_{diff} \geq 0$ , then  $[t_{e,1}; t_{e,d(\phi_1)} + \Delta_{gap} + \Delta_{future}] \subseteq T_1$  and the lemma holds for Base Case of the induction. Also, in Inductive Step of the induction, to calculate  $T_3$ , replace  $\Delta_{predict}$  by  $\Delta_{predict}^{min} + \Delta_{diff}$ . If  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$ , then  $T_3 = [t_{e,d(\phi_j)-1}; t_{e,d(\phi_j)} + \Delta_{gap} + \Delta_{future} + \Delta_{diff}]$  and  $T_4 = T_2 \cup T_3 = [t_{e,1}; t_{e,d(\phi_j)} + \Delta_{gap} + \Delta_{future} + \Delta_{diff}]$ . Since in this case  $\Delta_{diff} \geq 0$ , then  $[t_{e,1}; t_{e,d(\phi_j)} + \Delta_{gap} + \Delta_{future}] \subseteq T_4$  and the lemma holds for Inductive Step of the induction. So, Lemma 5.9 holds in this case.

**Case 2.**  $\Delta_{predict} < \Delta_{predict}^{min}$ . Consider the proof of Lemma 5.9. Then, in Base Case of the induction to calculate  $T_1$ , replace  $\Delta_{predict}$  by  $\Delta_{predict}^{min} + \Delta_{diff}$ . If  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$ , then  $T_1 = [t_{e,1}; t_{e,2} + \Delta_{gap} + \Delta_{future} + \Delta_{diff}]$ . As  $t_{e,2} = t_{e,d(\phi_1)}$ , we have  $T_1 = [t_{e,1}; t_{e,d(\phi_1)} + \Delta_{gap} + \Delta_{future} + \Delta_{diff}]$ . Since in this case  $\Delta_{diff} < 0$ , then  $T_1$  is a subset of  $[t_{e,1}; t_{e,d(\phi_1)} + \Delta_{gap} + \Delta_{future}]$  such that  $T_1$  is not equal to  $[t_{e,1}; t_{e,d(\phi_1)} + \Delta_{gap} + \Delta_{future}]$ . Therefore, the lemma does not hold for Base Case of the induction. So, Lemma 5.9 does not hold in this case.  $\square$

**Lemma 5.10.** *Let  $p_i$  and  $p_j$  be the processes defined in the time-limited completeness property of the neighbor detector. Then, at every round $_k$  such that  $k \geq 3$ , networkLocs of  $p_i$  contains accurate location predictions for both  $p_i$  and  $p_j$ , which are defined for the time interval  $[t_{e,1}; t_{e,k} + \Delta_{future}]$ .*

*Proof.* By Condition C1, we know that the execution that we consider for the proof is nice. By Corollary 5.1, which is a corollary of Lemma 5.3, we know that there exists at least one atomic phase which occurs before round $_k$ . Let  $\phi_i$  be the last atomic phase before round $_k$ . By Lemma 5.9, we know that once  $\phi_i$  terminates, networkLocs contains accurate location predictions for both  $p_i$  and  $p_j$  for the time interval  $[t_{e,1}; t_{e,d(\phi_i)} + \Delta_{gap} + \Delta_{future}]$ . By Lemma 5.7, we know that the maximum time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,k}$  is equal to  $\Delta_{gap}$ . Thus, in round $_k$ , networkLocs of  $p_i$  contains accurate location predictions for both  $p_i$  and  $p_j$  for the time interval  $[t_{e,1}; t_{e,k} + \Delta_{future}]$ .  $\square$

**Corollary 5.4.** *The minimum value of  $\Delta_{predict}$  for which Lemma 5.10 holds is  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$ .*

*Proof.* Consider the proof of Lemma 5.10. This proof relies on Lemma 5.9 which states that once  $\phi_i$  terminates, *networkLocs* of  $p_i$  contains accurate location predictions for both  $p_i$  and  $p_j$ , which are defined for the time interval  $[t_{e,1}; t_{e,d(\phi_i)} + \Delta_{gap} + \Delta_{future}]$ . This time interval is the minimum time interval required to prove Lemma 5.10, mainly because  $\Delta_{gap}$  is the maximum (and hence, the least upper bound) for the time duration between  $t_{e,d(\phi_i)}$  and  $t_{e,k}$ . Moreover, By Corollary 5.3, which is a corollary of Lemma 5.9, we know that  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$ , is the minimum value for which Lemma 5.9 holds. Therefore, based on the above arguments,  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$  is the minimum value for which Lemma 5.10 holds.  $\square$

**Theorem 5.1.** *The neighbor detector algorithm satisfies the time-limited completeness property.*

*Proof.* From Condition C6, we have  $t_c \geq t_{b,3}$ , which means that PRESENT( $t$ ) is called in *round<sub>k</sub>* such that  $k \geq 3$ . Also,  $t \geq t_{e,1}$  implies that the neighbor detection is not guaranteed for time instants before  $t_{e,1}$ . Thus, considering Condition C6, we can reformulate the theorem as follows. Let  $p_i$  and  $p_j$  be the two correct processes defined in the *time-limited completeness* property. Let PRESENT( $t$ ) be invoked at  $p_i$  in *round<sub>k</sub>* such that  $k \geq 3$ . If  $loc(p_j, t) \in Z(p_i, r_d, t)$  and  $t_{e,1} \leq t \leq t_c + \Delta_{future}$ , then  $p_j \in N(p_i, t)$  where  $t_c$  is the time when PRESENT( $t$ ) is called at  $p_i$ . For the proof, we proceed as follows. By Lemma 5.10, we know that at every *round<sub>k</sub>* such that  $k \geq 3$ , *networkLocs* of  $p_i$  contains accurate location predictions for both  $p_i$  and  $p_j$  for the time interval  $T_1 = [t_{e,1}; t_{e,k} + \Delta_{future}]$ . We know that in *round<sub>k</sub>*,  $t_c \in [t_{b,k}; t_{e,k}]$ . Thus, let  $T_2 = [t_{e,1}; t_c + \Delta_{future}]$  we have  $T_2 \subseteq T_1$ . Therefore, in *round<sub>k</sub>*, *networkLocs* of  $p_i$  contains accurate location predictions for both  $p_i$  and  $p_j$  for  $T_2$ . Since the algorithm guarantees the correct neighbor matching based on calculating the distance between the predicted locations (line 9), if  $loc(p_j, t) \in Z(p_i, r_d, t)$ , then  $p_j \in N(p_i, t)$  for  $\forall t \in T_2$ . So, the theorem holds.  $\square$

We next prove Theorem 5.2 according to which the algorithm correctly implements *the perfect accuracy* property. Roughly speaking, this property guarantees that no false neighbor detection occurs. Since for neighbor detection a process uses the location predictions stored in its *networkLocs*, to prove the theorem we basically show that: (1) the location predictions in *networkLocs* are indeed collected from the real nodes and are not created by the communication links; (2) the locations

are predicted accurately and (3) the algorithm correctly detects the neighbors of a process by calculating the distance between the predicted locations.

**Theorem 5.2.** *The neighbor detector algorithm satisfies the perfect accuracy property.*

*Proof.* Let  $p_i$  and  $p_j$  be the processes defined in the *perfect accuracy* property, according to the algorithm, if  $p_j \in N(p_i, t)$ , there exists a round during which  $p_i$  has received from a virtual node a *virtmsg* with a *collectedLocs* map such that a location prediction for key pair  $(p_j, t)$  exists in *virtmsg.collectedLocs*. The *integrity* property of the underlying broadcast (stated in Section 5.2.2) guarantees that the *virtmsg* is indeed sent by a virtual node. According to the algorithm, the *collectedLocs* map of the *virtmsg* is created by combining the *collectedLocs* of all virtual nodes in the system (line 45). These *collectedLocs* are encapsulated in *intervirtmsgs* and received from virtual nodes when a collect round terminates. The *integrity* property of the underlying broadcast guarantees that each *intervirtmsg* is indeed sent by a virtual node. The *collectedLocs* of *intervirtmsgs* contain location predictions that are collected by virtual nodes during the collect round. In the collect round, the location predictions are sent by real nodes in *locs* maps of *realmmsgs* (lines 32-34). The *integrity* property of the underlying broadcast guarantees that each *realmmsg* received from a real node is indeed sent by that real node. The *strong accuracy* property of the mobility predictor service (stated in Section 5.2.4) guarantees that the location predictions in *locs* maps are accurate. Moreover, the algorithm only detects  $p_j$  as a neighbor of  $p_i$  at time  $t$  if the distance between their predicted locations at  $t$  is less than or equal to  $r_d$  (line 9). Hence, if  $p_j \in N(p_i, t)$ , then  $loc(p_j, t) \in Z(p_i, r_d, t)$  and the theorem holds.  $\square$

**Theorem 5.3.** *The neighbor detector algorithm correctly implements the time-limited neighbor detector service.*

*Proof.* By Theorem 5.1, the algorithm guarantees the *time-limited completeness* property. By Theorem 5.2, the algorithm guarantees the *perfect accuracy* property. Hence, Theorem 5.3 holds.  $\square$

**Theorem 5.4.** *The minimum value of  $\Delta_{predict}$  for which neighbor detector algorithm correctly implements the time-limited neighbor detector service is  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$ .*

*Proof.* According to Theorem 5.3, the algorithm correctly implements the time-limited neighbor detector service. For the proof of Theorem 5.3 we used Theorem 5.1 which itself relies upon Lemma 5.10. By Corollary 5.4, which is a corollary of Lemma 5.10, we know that  $\Delta_{predict}^{min} = 2 \times \Delta_{scan} - 1 + \Delta_{gap} + \Delta_{future}$  is the minimum value for which Lemma 5.10 holds. Hence, Theorem 5.4 holds.  $\square$

**Corollary 5.5.** *In the general case of nice executions  $\Delta_{predict}^{min} = 8 \times \Delta_{scan} + \Delta_{unstable}^{max} + \Delta_{future} - 3$  and in a special case of nice executions where all virtual nodes are correct  $\Delta_{predict}^{min} = 4 \times \Delta_{scan} + \Delta_{future} - 1$ .*

*Proof.* In Theorem 5.4,  $\Delta_{predict}^{min}$  is defined as a function of  $\Delta_{gap}$ . From Lemma 5.6, we get that for nice executions in general  $\Delta_{gap} = 6 \times \Delta_{scan} - 2 + \Delta_{unstable}^{max}$ . However, by Corollary 5.2, which is a corollary of Lemma 5.6, we know that in a special case of nice executions where all virtual nodes are correct,  $\Delta_{gap} = 2 \times \Delta_{scan}$ . Thus, the corollary holds if in  $\Delta_{predict}^{min}$  we replace  $\Delta_{gap}$  with its value for each case.  $\square$

Since  $\Delta_{scan}$  and  $\Delta_{future}$  are positive integers and  $\Delta_{unstable}^{max}$  is a non-negative integer, from Corollary 5.5, we conclude that  $\Delta_{predict}^{min}$  in the case that all virtual nodes are correct is smaller than in the general case of nice executions. This result is very intuitive. In fact, in the general case of nice executions the location predictions should be long enough to do not expire during the unstable periods where virtual nodes crash. In the case where all virtual nodes are correct, there is no unstable period, therefore, the predictions do not need to be as long as in the general case.

### 5.4.6 Impact of Increasing the Number of Virtual Mobile Nodes on $\Delta_{predict}^{min}$

As discussed at the beginning of Section 5.4, one of our motivations for designing a neighbor detector algorithm based on multiple virtual mobile nodes, was that by growing the number of virtual mobile nodes (denoted by  $n$ ), we can reduce  $\Delta_{predict}$  required for the correctness of the algorithm. Thus, here we discuss the impact of increasing  $n$  on  $\Delta_{predict}^{min}$  (defined by Theorem 5.4). In order to do so, we first find an upper bound on  $\Delta_{predict}^{min}$  where the upper bound is a function of  $n$ . To find the upper bound we proceed as follows. In Corollary 5.5, which is the associated corollary of Theorem 5.4,  $\Delta_{predict}^{min}$  is expressed as a function of  $\Delta_{scan}$  (both in the general case of nice executions as well as in the case where all virtual nodes are correct). By

Eq. 5.5 of Section 5.4.4, we know that  $\Delta_{scan}$  is equal to  $\Delta_{scan}(v_i)$  where  $v_i$  can be any virtual node in the set of all virtual nodes in the system. Moreover, in Eq. 5.4 of Section 5.4.3 we have already defined an upper bound for  $\Delta_{scan}(v_i)$  where the upper bound is a function of  $n$ . Considering these facts, we find the following upper bound for  $\Delta_{predict}^{min}$ :

$$\Delta_{predict}^{min} < c'_1 + \frac{c'_2}{n} \quad (5.6)$$

where  $c'_1$  and  $c'_2$  are two constants defined as follows. Let  $c_1$  and  $c_2$  be the constants in Eq. 5.4 and whose values are defined by Eqs. 5.16 and 5.17 in Appendix 5.A. Then, for the general case of nice executions, we have:

$$c'_1 = \frac{8}{v_{avg}} \times c_1 - 3 + \Delta_{unstable}^{max} + \Delta_{future} \quad (5.7)$$

$$c'_2 = \frac{8}{v_{avg}} \times c_2 \quad (5.8)$$

And in the special case of nice executions where all virtual nodes are correct, we have:

$$c'_1 = \frac{4}{v_{avg}} \times c_1 - 1 + \Delta_{future} \quad (5.9)$$

$$c'_2 = \frac{4}{v_{avg}} \times c_2 \quad (5.10)$$

According to the Eq. 5.6, as  $n$  grows  $\Delta_{predict}^{min}$  decreases. Roughly speaking, this means that as the number of virtual nodes grows the algorithm requires smaller values of  $\Delta_{predict}$  to correctly implement the neighbor detector. However, note that as  $n$  approaches infinity, the right hand side of the equation approaches  $c'_1$  which is a constant. In practice, this means that if the number of virtual nodes is very large, adding more virtual nodes does not reduce any more the minimum value of  $\Delta_{predict}$  required for correctness of the algorithm.

## 5.5 Performance Discussion

In this section we discuss two topics related to the performance of our algorithm, namely its scalability with respect to the number of virtual nodes and the optimizations which can improve its performance.

### 5.5.1 Scalability with respect to the Number of Virtual Mobile Nodes

As discussed in Section 5.4.6, one of the advantages of our neighbor detector algorithm is that as  $n$  (i.e., the number of virtual nodes) grows the algorithm requires smaller values of  $\Delta_{predict}$  to correctly implement the neighbor detector. However, adding more virtual nodes also affects the consumption of resources such as energy and bandwidth. Communication is the main cause of energy and bandwidth consumption in a network executing the neighbor detector algorithm. Therefore, in this section we study the impact of increasing  $n$  on *the communication cost*. Note that since the algorithm is round-based and there exist an infinite number of rounds, we define the communication cost for one round, which can be measured by the number of broadcasts (via the underlying LocalCast service) in a round.

Let  $NOB(R_i)$  denote the number of broadcasts in a subregion  $R_i$  during a round. Basically,  $NOB(R_i)$  is an increasing function of  $\Delta_{scan}(v_i)$  (recall that  $v_i$  is the virtual node associated to  $R_i$ ). In fact, if  $\Delta_{scan}(v_i)$  increases,  $v_i$  scans its subregion longer. Consequently, more real nodes emulate  $v_i$ , which results in more broadcasts. In addition, if the round is a collect round, more real nodes send their predictions to  $v_i$  and if the round is a distribute round,  $v_i$  broadcasts the location predictions longer. Let *maximum broadcast rate* ( $mbr$ ) denote the maximum number of broadcasts per time unit in any subregion, we have:

$$NOB(R_i) \leq mbr \times \Delta_{scan}(v_i) \quad (5.11)$$

Note that  $mbr$  is a constant and independent of  $n$ . In fact,  $mbr$  is a function of the number of real nodes that are within a distance  $r_{mp}$  and/or  $r_{com}$  of the location of  $v_i$  per time unit. These are the real nodes that emulate  $v_i$  or send their location predictions to  $v_i$ . As in our study we assume that the real node density,  $r_{mp}$  and  $r_{com}$  are constant, then  $mbr$  is also a constant.

Let  $NOB(R)$  denote the number of broadcasts in the entire region  $R$  during a round (recall that  $R$  is the region for which the neighbor detection is implemented). Based on Eq. 5.11 and since the value of  $\Delta_{scan}$  is the same for all virtual nodes, we can find the following upper bound on  $NOB(R)$ :

$$NOB(R) \leq n \times mbr \times \Delta_{scan}(v_i) \quad (5.12)$$

Considering Eq. 5.12 and Eq. 5.4 of Section 5.4.3, which defines an upper bound for  $\Delta_{scan}(v_i)$ , we have:

$$NOB(R) < \frac{n \times mbr \times c_1}{v_{avg}} + \frac{mbr \times c_2}{v_{avg}} \quad (5.13)$$

where  $c_1$  and  $c_2$  are the constants of Eq. 5.4 and whose values are defined by Eqs. 5.16 and 5.17 in Appendix 5.A. Based on Eq. 5.13, we know that the number of broadcasts in a round has a complexity of  $\mathcal{O}(n)$ . This means that the communication cost of the algorithm scales linearly with the number of virtual mobile nodes. Although there exists no widely-accepted definition of scalability in the literature, it is usually assumed that an algorithm is scalable if its cost is less than  $\mathcal{O}(n^2)$  [47]. Therefore, we can say that the algorithm is scalable (in terms of communication cost) with respect to the number of virtual nodes.

## 5.5.2 Performance Optimization

It is beyond the scope of this paper to present a deployment of the neighbor detector algorithm in a real network. Here, we only discuss some ways in which the neighbor detector algorithm can be optimized for deployment purposes. It would be interesting to experiment with a real deployment to determine the extent to which these optimizations can be applied and whether they can effectively improve the performance of the algorithm.

Since the neighbor detector algorithm relies on virtual mobile nodes, optimizing the implementation of the virtual mobile nodes indirectly optimizes the neighbor detector algorithm. Thus, in the following we first discuss ways in which the implementation of the virtual mobile nodes can be optimized. We then discuss ways in which the neighbor detector algorithm can be directly optimized.

### 5.5.2.1 Optimizing the Implementation of the Virtual Mobile Nodes

In this paper we assumed that the virtual mobile nodes are implemented by the MPE algorithm sketched in Section 5.4.1. The MPE algorithm has a number of limitations. In particular, it requires significant amounts of communication and power consumption [14]. Moreover, it relies on the LocalCast service, a powerful local communication service which is both reliable and synchronous (i.e., it delivers the message after a bounded time interval). Below, we suggest two ways to deal with these limitations.

**Optimizing the MPE Algorithm.** In [14], Dolev et al. propose some optimizations for the MPE algorithm. These optimizations mainly reduce the number of message exchanges between the real nodes, which results in saving power as well. For instance, if a (temporary) leader is elected within a mobile point, and the leader initiates all the transitions for the replica, conflicting requests are avoided and power is saved. The interested reader can refer to [14] for more detail regarding these optimizations. Moreover, according to Dolev et al. the MPE algorithm can be correctly implemented using an underlying broadcast service that works in partially synchronous environments. They also propose some broadcast algorithms that can be used for implementing the MPE algorithm in such environments. The interested reader can refer to [15] for more detail.

**Using a Simpler Algorithm than the MPE algorithm to Implement the Virtual Mobile Nodes.** In [14], in addition to the MPE algorithm, Dolev et al. also describe an agent-based algorithm to implement the virtual mobile node abstraction. The algorithm uses a mobile agent, which is a special program (or as called in [14], a *dynamic process*) that jumps from one real node to another, moving in the direction specified by the trajectory function of the virtual mobile node. An agent *hitches a ride* with a host that is near to the specified location of the virtual mobile node. Compared to the MPE algorithm, this algorithm is simpler and more efficient (i.e., it requires less message exchanges and power consumption). However, it only achieves one of the goals of the design of a virtual mobile node i.e., the movement of the virtual mobile node can be predefined. In fact, the host of the agent is a single point of failure and therefore a virtual mobile node implemented by the agent-based algorithm is not robust. It seems likely that our neighbor detector algorithm can be correctly implemented (with some probability and under some conditions) even



if it uses the virtual nodes implemented by the agent-based algorithm. In this case the basic idea is that if the average time during which a host remains up after each recovery is known, then the agent can change its host just before the time when the host is likely to crash.

### 5.5.2.2 Optimizing the Neighbor Detector Algorithm

In addition to the implementation of the virtual nodes, the neighbor detector algorithm can also be directly optimized in many ways. Below we discuss some of these methods.

**Avoid Unnecessary Sharing and Distribution of Location Predictions.** In the current version of the neighbor detector algorithm, when a collect round terminates, each virtual mobile node shares the location predictions that it has collected with other virtual mobile nodes. However, sharing the location predictions may not be always necessary. In fact, users of mobile devices may remain for long periods of time in a subregion depending on their work habits, their movement speed, etc. In this case, the predicted locations of a real node are all in the same subregion where the real node resides at the moment of collection. Thus, if at the end of a collect round, for every virtual node  $v_i$ , the predicted locations that it has collected are all in  $R_i$  (as a reminder,  $R_i$  denotes the associated subregion of  $v_i$ ), then there is no need for virtual nodes to share what they have collected. Consequently, in the next distribute round, each virtual node only distributes the predictions that it has collected from its associated subregion. In the described example, the unnecessary distribution of location predictions is avoided indirectly, i.e., by avoiding the unnecessary sharing. However, it seems that the unnecessary distribution can also be avoided after the sharing. For instance, after the sharing, based on the location predictions, each virtual node  $v_i$  can identify the real nodes that will be in its associated subregion during the upcoming distribute round. Thus, if  $v_i$  can determine the location predictions which are never used for neighbor detection by these real nodes, it can avoid distributing them.

**Defining a Time Bound for the Detection of the Past Neighbors.** In the definition of the *time-limited completeness* property of the neighbor detector service (stated in Section 5.3.1.2), for simplicity's sake, we do not assume a time bound for the detection of the past neighbors. This implies that each real node should have

an unbounded buffer to store all the location predictions that it receives from the virtual nodes in the distribute rounds. However, for real deployments, based on the application requirements and the availability of the resources, a time interval down to which the past neighbors can be detected should be considered. In this way, the location predictions stored by a real node can be erased as soon as they become too old to be used for the neighbor detection.

**Defining Clusters for Real Nodes.** In the current version of the neighbor detector algorithm, each real node acts on its own and is responsible for sending and receiving messages to the virtual nodes. However, we can think of real nodes forming clusters such that within each cluster one or more real nodes, called the *gateway nodes*, are in charge of communicating with the virtual nodes. Based on how the clusters are defined, the other real nodes can communicate with these gateway nodes using the LocalCast service or a traditional MANET routing protocol. It seems that using clusters can reduce the scan path length of a virtual node (and consequently the scan duration or  $\Delta_{scan}$ ). In fact, in this case, during a scan a virtual node should only be in the transmission range of the gateway nodes instead of all real nodes. It remains an open question as to whether using clusters can also reduce the number of message exchange.

## 5.6 Related Work

For the related work, we consider three categories of algorithms, which are *neighbor detection algorithms*, *mobility-assisted algorithms* and *virtual mobile node-based algorithms*. Note that these categories are not disjoint. In particular, all virtual mobile node-based algorithms can also be categorized as mobility-assisted algorithms and some of them can also be categorized as neighbor detection algorithms. Our motivation for considering a category for virtual mobile node-based algorithms is to be able to discuss in detail how these algorithms use virtual mobile nodes to achieve their goals and to compare their approach with the approach used by our algorithm.

### 5.6.1 Neighbor Detection Algorithms

In ad hoc networks neighbor detection is usually studied as a building block for applications such as routing, leader election, group management and localization. The majority of the existing neighbor detection algorithms belong to the *hello protocols* family [37; 50; 17; 28; 1; 26; 24; 4; 6]. They are based on the *basic hello protocol* first described in *Open Shortest Path First (OSPF)* routing protocol [38], which works as follows: each node in the network periodically sends *hello* messages to announce its presence to close nodes, and maintains a neighbor set. The sending frequency is denoted by  $f_{hello}$ . If a *hello* message is not received from a neighbor for a predefined amount of time, then that neighbor is discarded from the neighbor set. The problem with this approach is that if  $f_{hello}$  is too low (with respect to the speed of the nodes), then the neighbor set becomes quickly obsolete. On the other hand, if it is too high, the neighbor set remains up to date but it causes a significant waste of communication bandwidth and energy [26]. However, finding the optimal  $f_{hello}$  is not obvious and the existing solutions cannot ideally solve this problem. Moreover, contrary to our neighbor detector algorithm, the *hello protocols* usually provide only the set of current neighbors and they do not satisfy any formal guarantees.

Although, the *hello protocols* comprise the majority of the existing neighbor detection algorithms for ad hoc networks, in the literature there exist also the schemes that use different approaches than the *hello* broadcast for neighbor detection [11; 12]. For instance, in [12] Cornejo et al. define a reliable neighbor detection abstraction that establishes links over which message delivery is guaranteed. They present two region-based neighbor detection algorithms which implement the abstraction with different link establishment guarantees. The algorithms are implemented on top of a Medium Access Control (MAC) layer which provides upper bounds on the time for message delivery. The main idea behind the first algorithm is that a node sends a *join* message some time after entering a new region to establish communication links. It also sends a *leave* message some time before leaving a region to inform the other nodes so that they can tear down their corresponding link with that node. To guarantee that these notification messages reach their destination despite the continuous motion of nodes, the authors define the time limits for a node to send the *join* and the *leave* messages. These time limits are obtained using the timing guarantees of the underlying MAC layer. Since a node should send a *leave* message some time before it actually leaves a region, the algorithm assumes that a node's

trajectory function is known to that node with enough anticipation to communicate with other nodes before leaving the region. The first algorithm does not guarantee the communication links when nodes are moving quickly across region boundaries. Thus, the authors introduce a second algorithm. In this new algorithm they apply a technique which overlays multiple region partitions, associating with each region partition an instance of the first algorithm. The output of each instance is then composed such that it guarantees the communication links even when nodes are moving across region boundaries. The approach applied in [12] for neighbor detection is interesting because it uses a relatively lower number of message broadcast compared to the *hello protocols*. Similarly to our work, this approach also uses the knowledge of nodes about their future locations for the neighbor detection. However, contrary to our work, no future neighbor detection is defined and only the current neighbor detection is guaranteed.

The time-limited neighbor detector service implemented in this paper is first introduced in our previous work in [5]. To the best of our knowledge, it is the only neighbor detector service for ad hoc networks that detects not only the current neighbors of a node but also its future neighbors. In addition to the definition of the neighbor detector, in [5] we also proposed a simple but limited algorithm that implements the neighbor detector using a single virtual mobile node. In Section 5.6.3, which is dedicated to the virtual mobile node-based algorithms, we will describe this simple algorithm in more detail and compare it with the algorithm introduced in the present work.

## 5.6.2 Mobility-assisted Algorithms

In the literature of mobile ad hoc networks, node mobility is leveraged for different purposes e.g., to improve security [8], increase network capacity [19] or locating nodes [20]. In particular, there exist various algorithms that take advantage of mobility for routing purposes in sparse mobile ad hoc networks [48; 56; 42; 18; 32; 23; 46; 43; 9; 31; 53; 55; 54; 2; 41].<sup>7</sup> In a sparse mobile ad hoc network, node deployment is sparse. Therefore, nodes may not be in the transmission range of each other for long periods of time. Several routing algorithms for this type of networks

---

<sup>7</sup> Since a sparse mobile ad hoc network is a specific type of delay tolerant networks (DTNs), in the literature, some of these algorithms are presented as routing algorithms for DTNs.

use the *store-carry-forward* model, according to which the nodes in the network forward a message from the source node to the destination node in one or many hopes, such that, once a relay node receives the message, it stores and carries the message until it has a chance to forward it to another node [53]. Thus, although in such networks an end-to-end path between a source and a destination may never exist at a given time, the *store-carry-forward* model uses node mobility to provide paths between the source and the destination overtime. The famous examples of the algorithms that use the *store-carry-forward* model, are the *epidemic routing* algorithms [42; 18; 32; 23; 46; 43] which are based on the original *epidemic routing* algorithm of Vahdat et al. [48]. The key idea behind this algorithm is that, similarly to the spread of infectious diseases, a node carrying a message forwards it to all other nodes that it meets and which do not have a copy of the message [56]. The *epidemic routing* algorithms exploit the inherent node mobility. In [22], Hatzis et al. introduce the concept of *compulsory* protocols, which require a subset of the mobile nodes to move in a *pre-specified* manner. The motivation behind the design of *compulsory* protocols is that if mobile nodes moved in a programmable way, algorithms could take advantage of motion, performing even more efficiently than in static networks [14]. In [22], Hatzis et al. present an efficient compulsory protocol for leader election. Furthermore, Chatzigiannakis et al. [9] and Li et al. [31] propose simple and efficient routing algorithms based on the idea of compulsory protocols.

Among the mobility-assisted algorithms that do not use virtual mobile nodes, the closest algorithms to our work are the *message ferrying* (MF) algorithms [53; 55; 54; 2]. MF algorithms are the routing algorithms for sparse ad hoc networks, which use moving entities called *message ferries* (or *ferries* for short) for carrying messages between nodes. Ferries travel through the network and communicate with nodes using a single-hop broadcast scheme. Similarly to a virtual mobile node's path, the route through which a ferry moves is programmed and usually known to all nodes in the network. However, contrary to a virtual mobile node, a ferry is, in fact, a real node which has no (or less) resource constraints (in terms of energy consumption or buffer size) compared to other nodes in the network. Ferries are used in different applications. For instance, in a disaster scene where the existing infrastructure is unusable, airplanes or vehicles can be used as ferries to transport data between users in separated areas. In sensor networks with limited power supplies, mobile entities such as robots can be used as ferries to collect data from sensors and thus,

reduce the part of sensor energy that is consumed for communication purposes [55].<sup>8</sup> The MF algorithms usually aim at satisfying some guarantees in terms of network throughput, average message delay or energy consumption. Therefore, the routes of the ferries should be designed so that such guarantees can be achieved. In [53; 55] the ferry route design problem is considered for a network where regular nodes<sup>9</sup> are stationary and their locations are *a priori* known. Thus, the ferry route design problem is basically considered as a variation of the *traveling salesman problem* (TSP) and is solved by applying some TSP algorithms on the locations of the regular nodes. Obviously, the solutions proposed in [53; 55] are limited since they can only be applied to stationary networks where the locations of regular nodes are *a priori* known. In [54], a network composed of mobile regular nodes is considered. Since regular nodes are mobile, to ensure the meeting between the ferries and the regular nodes, two approaches are proposed where each approach is presented by one MF algorithm. In the Node-Initiated MF (NIMF) algorithm, ferries move around the deployed area according to routes known to the regular nodes. When a regular node has a message to send or receive, it moves close to a ferry and communicates with it. The problem with this approach is that it disrupts the normal movement of the regular nodes for the purpose of communication with the ferries. In the Ferry-Initiated MF (FIMF) algorithm, the ferries move to meet the regular nodes at their requests. Thus, when a regular node wants to send a message to another regular node or receive a message, it generates a *service request* (which is a control message encapsulating the location of the regular node) and transmits it to a chosen ferry using a long-range radio. Upon the reception of a *service request*, the ferry will adjust its trajectory to meet up with the regular node and exchange messages using short range radios. Note that since the location of the regular node can change after sending the *service request*, the regular node occasionally transmits *location updates* (which are also control messages) using long range radio to notify the ferry of the changes in its location. The main problem with this approach is that it requires the use of long-range radio which might not always be feasible or desirable. In [2], a ferry route design algorithm called Optimized Way Points (OPWP) is proposed

---

<sup>8</sup> In the literature, there exists also the notion of *Data MULEs* (Mobile Ubiquitous LAN Extensions) [41]. Message ferries and Data MULEs are somehow synonymous. The main difference between them is that the movement of Data MULEs is random [2; 43; 54].

<sup>9</sup> In the MF algorithms, the nodes in the network which are not ferries are referred to as *regular nodes*. We also adopt this term while discussing MF algorithms.

for a network where regular nodes are mobile. OPWP only guarantees probabilistic meetings between the ferries and the regular nodes, that is, it ensures that every time a ferry traverses its route, it meets every regular node with a certain minimum probability. A ferry route found by OPWP comprises an ordered set of way-points and waiting times at these way-points that are chosen carefully based on the mobility model of the regular nodes. More precisely, to choose the set of way-points for a ferry route, OPWP requires that for every regular node  $p_i$  and every way-point  $s$  in the deployment area, the probability of the meeting between the ferry and  $p_i$  when the ferry moves as well as the probability of the meeting between the ferry and  $p_i$  when the ferry waits at  $s$  to be known. Thus, the main problem with the approach used by OPWP is that to find the routes of the ferries, not only the mobility model of the regular nodes should be *a priori* known but also the mobility model should be such that the above mentioned probabilities can be determined from it.

As we have described, the MF algorithms require some nodes in the network (basically the ferries) to move in a controlled, programmed way (note that in some MF algorithms such as in the NIMF algorithm, not only the ferries but also the regular nodes should adjust their movement for the communication purposes) whereas our work is based on virtual mobile nodes which only use the real nodes to emulate the virtual mobile node and does not require them to move in a programmed way. Not requiring the real nodes to move in a programmed way is preferable especially in the PBM applications (i.e., the target applications for our work) where the wireless devices are used by ordinary people who are not amenable to following instructions as to where their devices may travel. Moreover, as described above, the approaches proposed in the literature to compute the ferry routes, are either limited (e.g., require stationary regular nodes with known locations) or complicated (e.g., require sending control messages using long range radio or require some probabilities to be determined from the mobility model of the regular nodes). On the contrary, our approach for finding the scan path of virtual mobile nodes does not require a stationary network and is simple, since it only requires finding the covering centers using a hexagonal tessellation algorithm.

In the next section we discuss a special type of mobility-assisted algorithms, i.e., *the virtual mobile node-based* algorithms and compare them to our algorithm.

### 5.6.3 Virtual Mobile Node-based Algorithms

The idea of using virtual mobile nodes to facilitate the design of algorithms for mobile ad hoc networks was first introduced by Dolev et al. in [14]. The virtual mobile node abstraction design was inspired by idea of the compulsory protocols of Hatzis et al. [22] (we have already discussed the compulsory protocols in Section 5.6.2). Note however that, contrary to the compulsory protocols, the virtual mobile node-based algorithms do not require a set of real nodes to move in a programmable way but they rather require the virtual mobile nodes, emulated by the real nodes, to move in a programmable way.

In [15], several basic algorithms that use virtual mobile nodes to solve various problems are briefly presented. These algorithms address the problems such as routing, collecting and evaluating data, group communication and atomic memory in mobile ad hoc networks. Similarly to the present work, some of these algorithms use several virtual nodes which regularly exchange information between each other. However, contrary to the present work, no explicit properties for the trajectory functions of the virtual nodes are defined to guarantee the meeting and communication between them.

In our previous work [5], we presented an algorithm that implements the time-limited neighbor detector service with the same guarantees defined in the present work. Similarly to the present work, real nodes have access to a mobility predictor service and can predict their locations up to  $\Delta_{predict}$  in the future. However, the algorithm uses only a single virtual mobile node that travels through the network, collects location predictions and distributes neighbor detection-related information. Thus, in order to stay correct, the algorithm requires greater  $\Delta_{predict}$  values as the map size grows. Another drawback of the algorithm presented in [5] is that it does not tolerate the failure of the virtual mobile node. These drawbacks are the main motivations for the present work. Finally, note that the present work is an extension of the work published as a conference paper in [7].

## 5.7 Conclusion

We have introduced an algorithm that implements the time-limited neighbor detector service for MANETs using  $n = 2^k$  virtual mobile nodes where  $k$  is a non-negative



integer. We proved that our algorithm is correct under certain conditions. In particular, we showed that our algorithm is correct for a category of executions, called *nice executions*, which basically correspond to the executions of the algorithm in periodically well-populated regions. We also defined the minimum value of  $\Delta_{predict}$  for which the algorithm is correct in different cases of nice executions.

Compared to the previously proposed algorithm [5], which uses a single virtual mobile node, our algorithm has two advantages: (1) it tolerates the failure of one to all virtual mobile nodes; (2) as the number of virtual mobile nodes grows, it remains correct with smaller values of  $\Delta_{predict}$ . This feature makes the real-world deployment of the neighbor detector easier since with the existing prediction methods, location predictions usually tend to become less accurate as  $\Delta_{predict}$  increases. We showed that the cost of our algorithm (in terms of communication) scales linearly with the number of virtual mobile nodes. We also proposed a set of optimizations which can be used for the real-world deployment of our algorithm.

To the best of our knowledge, this is the first work that uses multiple virtual mobile nodes to implement a neighbor detector service in MANETs. Another novelty of our work, is the definition of explicit properties for the scan paths of virtual nodes and then presenting a way to compute the scan paths. As shown in the paper, the scan paths are defined so that they guarantee a full collection and distribution of predictions in each subregion as well as the coordination between the virtual mobile nodes. Thus, we believe that the approach used in this paper to define the scan paths can be used for implementing other virtual mobile node-based algorithms such as virtual mobile node-based routing algorithms.

As a potential future work, we consider a real-world deployment of our neighbor detector algorithm. In fact, we are currently developing, *ManetLab*, a modular and configurable software framework for creating and running testbeds to evaluate MANET-specific protocols [49]. Once the neighbor detector algorithm is deployed on ManetLab, we can perform the following:

- compare our theoretical results with the results obtained from the real deployment. In fact, as shown by a quantitative analysis in the paper, when  $n$  (i.e., the number of virtual nodes) grows, the neighbor detector algorithm requires smaller values of  $\Delta_{predict}$  to correctly implement the neighbor detector and at the same time its communication cost grows as  $\mathcal{O}(n)$ . Thus, it would be interesting to validate our theoretical results using the real deployment and determine up to which value of  $n$  the utility of the algorithm outweighs its cost;

- deploy some other famous neighbor detection algorithms (mentioned in Section 5.6.1) and compare their performance with the algorithm in the present work. Note that since other neighbor detection algorithms (except the algorithm in our previous work [5]) only detect current neighbors the comparison will only be limited to current neighbor detection;
- apply the optimizations proposed in Section 5.5.2 on the deployment and determine the extent to which these optimizations can be applied and whether they can effectively improve the performance of the algorithm.

The neighbor detector algorithm presented in this paper is designed for periodically well-populated regions. Thus, another issue which can be investigated as future work is to implement the neighbor detector for less populated regions. In order to do so, we can think of using another type of virtual nodes called *autonomous virtual mobile nodes* [16]. This type of virtual nodes can move autonomously, choosing to change their path based on their own state and inputs from the environment. For instance, if the area in their paths appears deserted, they can change their path to the more populated areas.

## 5.A Appendix: Finding an Upper Bound for the Scan Path Length of a Virtual Mobile Node

According to Eq. 5.1 of Section 5.4.3, the scan path length of a virtual node  $v_i$  or  $L_{scan-path}(v_i)$  can be calculated as:

$$L_{scan-path}(v_i) = (NOC(R_i) - 1) \times \sqrt{3}r_{com}$$

where  $NOC(R_i)$  denotes the number of covering centers for subregion  $R_i$ . We can find an upper bound on  $NOC(R_i)$  by counting the number of lattice points that are associated to the tessellation of  $R_i$ . In mathematics and group theory, a two dimensional *lattice* is a discrete subgroup of  $\mathbb{R}^2$  which spans the vector space of  $\mathbb{R}^2$ . In a regular hexagonal tessellation, the centers and the vertices of the hexagons form a two dimensional lattice called the *hexagonal lattice*. Let  $NOL_{disk}$  denote the number of hexagonal lattice points within a disk centered at the origin of the plane (which is  $l_{map-center}$  in our case). Then,  $NOL_{disk}$  can be calculated using Eq. 5.14

where  $r$  is the radius of the disk and  $d$  is the distance between the closest lattice point pairs [30].

$$\begin{aligned}
NOL_{disk}(r, d) = & \sum_{x=-\lfloor \frac{r}{d\sqrt{3}} \rfloor}^{\lfloor \frac{r}{d\sqrt{3}} \rfloor} (2\lfloor \sqrt{\frac{r^2}{d^2} - 3x^2} \rfloor + 1) + \\
& \sum_{x=\frac{1}{2}-\lfloor \frac{r}{d\sqrt{3}} + \frac{1}{2} \rfloor}^{\lfloor \frac{r}{d\sqrt{3}} + \frac{1}{2} \rfloor - \frac{1}{2}} 2\lfloor \sqrt{\frac{r^2}{d^2} - 3x^2 + \frac{1}{2}} \rfloor
\end{aligned} \tag{5.14}$$

Obviously, a covering center is also a hexagonal lattice point. Thereby, if we want to calculate an upper bound for the number of covering centers of  $R_i$  using the number of the lattice points, we should take into account the limit cases i.e., where a covering center is at the boundary or out of  $R_i$ . Since the circumradius of a hexagon is  $r_{com}$ , we know that a covering center cannot be located at a distance greater than  $r_{com}$  from a boundary of  $R_i$ . Thus, we calculate the number of lattice points for an extended region  $R'_i$  made from  $R_i$ . The extension is performed by increasing the arc of  $R_i$  by  $2r_{com}$  (i.e., adding  $r_{com}$  at each end of the arc) and by increasing the radius of  $R_i$  by  $r_{com}$ . Therefore,  $R'_i$  has a radius of length  $r_{map} + r_{com}$  and a central angle  $\theta' = \frac{2r_{com}}{r_{map} + r_{com}} + \frac{2\pi}{n}$ . Let  $NOL_{region}(R'_i)$  denote the number of lattice points in  $R'_i$ , we have:

$$\begin{aligned}
NOL_{region}(R'_i) &= NOL_{disk}(r_{map} + r_{com}, r_{com}) \times \frac{\theta'}{2\pi} \\
&= NOL_{disk}(r_{map} + r_{com}, r_{com}) \times \\
&\quad \left( \frac{2r_{com}}{r_{map} + r_{com}} + \frac{2\pi}{n} \right) \times \frac{1}{2\pi}
\end{aligned} \tag{5.15}$$

As described before,  $NOC(R_i) < NOL_{region}(R'_i)$ , thus, considering this fact and Eq. 5.1 of Section 5.4.3 and Eq. 5.15, we have:

$$L_{scan-path}(v_i) < c_1 + \frac{c_2}{n}$$

where  $c_1$  and  $c_2$  are two constants defined below.

$$c_1 = \sqrt{3}r_{com} \times \left( \frac{r_{com} \times NOL_{disk}(r_{map} + r_{com}, r_{com})}{\pi \times (r_{map} + r_{com})} - 1 \right) \quad (5.16)$$

$$c_2 = \sqrt{3}r_{com} \times NOL_{disk}(r_{map} + r_{com}, r_{com}) \quad (5.17)$$

## References

- [1] M. Bakht, M. Trower, and R. Kravets. Searchlight: helping mobile devices find their neighbors. In *ACM SIGOPS Oper. Syst.*, vol. 45, no. 3, pp. 71–76, 2012.
- [2] M. M. Bin Tariq, M. Ammar and E. Zegura, Message ferry route design for sparse ad hoc networks with mobile nodes, In *Proc. ACM MobiHoc'06*, pp. 37–48, 2006.
- [3] B. Bostanipour, B. Garbinato and A. Holzer, Spotcast – A communication abstraction for proximity-based mobile applications, In *Proc. IEEE NCA'12*, pp. 121–129, 2012.
- [4] B. Bostanipour and B. Garbinato, Improving neighbor detection for proximity-based mobile applications, In *Proc. IEEE NCA'13*, pp. 177–182, 2013.
- [5] B. Bostanipour and B. Garbinato, Using virtual mobile nodes for neighbor detection in proximity-based mobile applications, In *Proc. IEEE NCA'14*, pp. 9–16, 2014.
- [6] B. Bostanipour and B. Garbinato, Effective and efficient neighbor detection for proximity-based mobile applications, In *Elsevier Computer Networks Journal*, vol. 79, pp. 216–235, 2015.
- [7] B. Bostanipour and B. Garbinato, Neighbor detection based on multiple virtual mobile nodes, In *Proc. IEEE PDP'16*, pp. 322–327, 2016.
- [8] S. Capkun, J.-P. Hubaux and L. Buttyan, Mobility helps security in ad hoc networks, In *Proc. ACM MobiHoc'03*, pp. 46–56, 2003.
- [9] I. Chatzigiannakis, S. E. Nikolettseas and P. G. Spirakis, An efficient communication strategy for ad-hoc mobile networks, In *Proc. DISC'01*, pp. 285–299, 2001.
- [10] C. Cheng, R. Jain and E. van den Berg, Location prediction algorithms for mobile wireless systems, In *Handbook of Wireless Internet*, M. Illyas and B. Furht, Eds. CRC Press, 2003.

- [11] A. Cornejo, S. Viqar, J. L. Welch, Reliable neighbor discovery for mobile ad hoc networks, In *Proc. DIALM-PODC'10*, pp. 63-72, 2010.
- [12] A. Cornejo, S. Viqar and J. L. Welch, Reliable neighbor discovery for mobile ad hoc networks, In *Ad Hoc Networks*. 12 (January 2014), pp. 259–277, 2014.
- [13] TMT. Do and D. Gatica-Perez, Contextual conditional models for smartphone-based human mobility prediction, In *Proc. ACM UbiComp'12*, pp. 163-172, 2012.
- [14] S. Dolev, S. Gilbert, N. A. Lynch, E. Schiller, A. A. Shvartsman, J. L. Welch, Virtual mobile nodes for mobile ad hoc networks, In *Proc. DISC'04*, pp. 230–244, 2004.
- [15] S. Dolev, S. Gilbert, N. A. Lynch, E. Schiller, A. A. Shvartsman, J. L. Welch, Virtual mobile nodes for mobile ad hoc networks, Tech Report LCS-TR-937, MIT, 2004.
- [16] S. Dolev, S. Gilbert, E. Schiller, A. A. Shvartsman, J. L. Welch: Autonomous virtual mobile nodes. In *DIALM-POMC*, pp. 62-69, 2005.
- [17] P. Dutta and D. Culler, Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proc. ACM SenSys'08*, 2008.
- [18] R. Groenevelt, P. Nain, and G. Koole, The message delay in mobile ad hoc networks, In *Elsevier Journal of Performance Evaluation*, vol. 62, pp. 210–228, 2005.
- [19] M. Grossglauser and D. Tse, Mobility increases the capacity of ad hoc wireless networks, In *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 477–486, 2002.
- [20] M. Grossglauser and M. Vetterli, Locating nodes with EASE: mobility diffusion of last encounters in ad hoc networks. In *Proc. IEEE INFOCOM'03*, 2003.
- [21] B. Grunbaum and G.C. Shephard, Tilings and patterns, New York: W. H. Freeman, 1990.
- [22] K. P. Hatzis, G. P. Pentaris, P. G. Spirakis, V. T. Tampakas and R. B. Tan, Fundamental control algorithms in mobile networks, In *Proc. ACM SPAA'99*, 1999.
- [23] Z. J. Haas and T. Small, A new networking model for biological applications of ad hoc sensor networks, In *IEEE/ACM Trans. Netw.*, vol. 14, pp. 27–40, 2006.
- [24] D. He, N. Mitton, and D. Simplot-Ryl, An energy efficient adaptive HELLO algorithm for mobile ad hoc networks, In *Proc. ACM MSWiM'13*, pp. 65–72, 2013.
- [25] iGroups. <http://tinyurl.com/y9cwvmx>.

- [26] F. Ingelrest, N. Mitton, and D. Simplot-Ryl, A turnover based adaptive hello protocol for mobile ad hoc and sensor networks, In *Proc. MASCOTS'07*, pp. 9–14, 2007.
- [27] I. M. Isaacs, *Geometry for College Students*, American Mathematical Society, 2009.
- [28] A. Kandhalu, K. Lakshmanan and R. R. Rajkumar, U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol, In *IPSN'10*, pp. 350–361, 2010.
- [29] B. Korte and J. Vygen, *Combinatorial optimization: theory and algorithms* (5 ed.), Springer, 2012.
- [30] P. Lax and R. Phillips, The asymptotic distribution of lattice points in Euclidean and non-Euclidean spaces. In *J. Funct. Anal.*, 46(3), pp. 280–350, 1982.
- [31] Q. Li and D. Rus, Sending messages to mobile users in disconnected ad-hoc wireless networks, In *Proc. MobiCom'00*, 2000.
- [32] A. Lindgren, A. Doria and O. Schelen, Probabilistic routing in intermittently connected networks, In *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, pp. 19–20, 2003.
- [33] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
- [34] LocoPing. <http://www.locoping.com/>
- [35] LoKast. <http://www.lokast.com/>
- [36] Local Multiplayer Apps. <http://appcrawlr.com/ios-apps/best-apps-local-multiplayer>
- [37] M. J. McGlynn and S. A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proc. ACM MobiHoc'01*, pp. 137–145, 2001.
- [38] J. Moy, OSPF – Open Shortest Path First, RFC 1583, 1994.
- [39] N. B. Priyantha, A. Chakraborty and H. Balakrishnan, The cricket location-support system, In *Proc MobiCom'00*, 2000.
- [40] S. Scellato, M. Musolesi, C. Mascolo, V. Latora and A. T. Campbell, Nextplace: a spatio-temporal prediction framework for pervasive systems, In *Proc. Pervasive'11*, pp. 152–169, 2011.
- [41] R. Shah, S. Roy, S. Jain and W. Brunette, Data MULEs: modeling a three-tier architecture for sparse sensor networks, In *Elsevier Ad Hoc Networks Journal*, vol. 1, pp. 215–233, 2003.

- [42] G. Sharma, R. Mazumdar and N. Shroff, Delay and capacity tradeoffs in mobile ad hoc networks: a global perspective, In *IEEE/ACM Trans. Netw*, vol. 15, pp. 981–992, 2007.
- [43] T. Small and Z. J. Haas, Resource and performance tradeoffs in delay-tolerant wireless networks, In *Proc. ACM WDTN'05*, pp. 260–267, 2005.
- [44] L. Song, D. Kotz, R. Jain and X. He, Evaluating location predictors with extensive Wi-Fi mobility data, In *Proc. IEEE INFOCOM'04*, pp. 1414–1424, 2004.
- [45] L. Song, U. Deshpande, U.C. Kozat, D. Kotz and R. Jain, Predictability of WLAN mobility and its effects on bandwidth provisioning. In *Proc. IEEE INFOCOM'06*, 2006.
- [46] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, Spray and wait: an efficient routing scheme for intermittently connected mobile networks, In *Proc. ACM WDTN'05*, pp. 252–259, 2005.
- [47] N. Suzuki, Shared memory multiprocessing, MIT Press, Cambridge, MA, USA, 1992.
- [48] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Tech Report CS-200006, Duke University, 2000.
- [49] F. Vessaz, B. Garbinato, A. Moro and A. Holzer, Developing, deploying and evaluating protocols with ManetLab. In *Proc. NETYS'13*, pp. 89–104, 2013.
- [50] G. Wattenhofer, G. Alonso, E. Kranakis and P. Widmayer, Randomized protocols for node discovery in ad hoc, single broadcast channel networks, In *Proc. IPDPS'03*, 2003.
- [51] Waze. <https://www.waze.com/en/>
- [52] WhosHere. <http://whoshere.net/>
- [53] W. Zhao and M. Ammar, Message ferrying: proactive routing in highly-partitioned wireless ad hoc networks, In *Proc. IEEE FTDCS'03*, pp. 308–314, 2003.
- [54] W. Zhao, M. Ammar and E. Zegura, A message ferrying approach for data delivery in sparse mobile ad hoc networks, In *Proc. ACM MobiHoc'04*, pp. 187–198, 2004.
- [55] W. Zhao, M. Ammar and E. Zegura, Controlling the mobility of multiple data transport ferries in a delay-tolerant network, In *Proc. IEEE INFOCOM'05*, vol. 2, pp. 1407–1418, 2005.

- [56] X. Zhang, G. Neglia, J. Kurose and D. Towsley, Performance modeling of epidemic routing, In *Elsevier Computer Networks Journal*, vol. 51, pp. 2867–2891, 2007.



# Chapter 6

## Conclusion

In this thesis we introduced a set of programming abstractions and algorithms that can be used for building PBM applications in MANETs. We started by identifying *proximity-based durable broadcast* and *proximity-based neighbor detection* as main requirements of PBM applications. In the first part of this thesis we proposed abstractions and algorithms for proximity-based durable broadcast, while in the second part, we proposed abstractions and algorithms for proximity-based neighbor detection. In this chapter, we first review the main contributions of this thesis and then discuss possible future research directions.

### 6.1 Contributions

**Spotcast and scoped broadcast abstractions.** We introduced spotcast, a new communication abstraction for proximity-based durable broadcast in MANETs. Spotcast enables a node to disseminate a message for a given time duration to all nodes located within a given range. Spotcast comes in three variants, namely *timely spotcast*, *eventual spotcast* and *exhaustive spotcast*. These variants differ in their timing guarantees for message delivery, which are defined based on the requirements of different PBM applications. Accordingly, each spotcast variant can be used to build a specific type of PBM applications. We also introduced *scoped broadcast*, a communication abstraction that enables a node to disseminate a message to all nodes located within a given range. Scoped broadcast comes in two variants: a *timely scoped broadcast*, which guarantees a synchronous message delivery, and a *fair-loss scoped broadcast*, which is an asynchronous variant. We presented a *generic* algorithm that implements the three spotcast variants using different scoped broadcast

variants and different types of message buffers. We also presented a proof of correctness for our algorithm. In addition, we discussed how scoped broadcast variants can be implemented in single-hop and multi-hop cases.

**Adjusting the parameters of the hello protocols for proximity-based neighbor detection.** We identified the transmission power and the broadcast interval as the main parameters of the hello protocols, which influence the effectiveness and the efficiency of neighbor detection. Effectiveness refers to the degree to which the detection is successful and efficiency refers to the degree to which the detection is energy saving. We evaluated the impact of these parameters on the effectiveness and efficiency of the hello protocols in three urban environments namely, *indoor with hard partitions* (corresponding to offices with thick walls), *indoor with soft partitions* (corresponding to indoor exhibitions with temporary partitions) and *outdoor urban areas* (corresponding to a music festival in downtown). These are the typical environments where PBM applications are used. Our evaluations were based on realistic simulations. For one thing, we used a faithful simulation of the *802.11a* technology for communication between nodes and we assumed a probabilistic radio propagation model for urban environments. Furthermore, we calculated the energy consumption using the real specification of typical smartphones. We identified the most effective strategy and the most efficient strategy in each environment, where a strategy refers to a pair of the transmission power and the broadcast interval. We showed that the most effective strategy is not the same as the most efficient strategy in any environment. In fact, in all environments, there is a conflict between effectiveness and efficiency such that the most effective strategy is usually not very efficient and the most efficient strategy is not always very effective. However, the conflict becomes less severe as the environment becomes less obstructed. We then proposed an approach to make a tradeoff between effectiveness and efficiency. Accordingly, we identified the tradeoff strategy in each environment and we showed that it had a relatively good effectiveness and efficiency compared to other strategies. Our results can be used as a basis to design adaptive neighbor detection algorithms for urban environments. Such algorithms can adapt the transmission power and broadcast interval based on environment and application requirements on effectiveness and efficiency.

**The time-limited neighbor detector abstraction and its implementations based on virtual mobile nodes.** We introduced a new abstraction for proximity-based neighbor detection called the *time-limited neighbor detector*. This abstraction

enables a node to detect its neighbors in the past, present and up to some bounded time interval in the future. It has a *completeness* property that guarantees to detect all past, present and future neighbors (up to some bounded time interval). It has also an *accuracy* property that guarantees that no false detection occurs. To the best of our knowledge, this is the first neighbor detector service that detects the future neighbors of a node in MANETs. We introduced two algorithms that implement the time-limited neighbor detector abstraction based on the notion of virtual mobile nodes already presented in the literature. To the best of our knowledge, these are the first algorithms that use virtual mobile nodes for neighbor detection in MANETs. More precisely, we introduced:

- A simple but limited algorithm based on a single virtual mobile node. Each real node has access to a mobility predictor service that accurately predicts its locations up to some bounded time interval  $\Delta_{predict}$  in the future. Thus, the virtual mobile node travels through the network, collects the location predictions of real nodes and distributes the neighbor lists, which it creates based on the collected location predictions, to real nodes. The algorithm is correct if the virtual mobile node is correct.
- A more general algorithm that uses  $n = 2^k$  virtual mobile nodes, where  $k$  is a non-negative integer. The algorithm implements the neighbor detector for real nodes located in a circular region. Each real node has access to the same mobility predictor service already defined for the single virtual mobile node-based algorithm. The key idea of the algorithm is that the virtual mobile nodes regularly collect location predictions of real nodes from different subregions, meet to share what they have collected with each other and then distribute the collected location predictions to real nodes. Thus, each real node can find its neighbors at current and future times based on the distributed location predictions. It can also store the location predictions so it can be queried about its past neighbors. We showed that the algorithm is correct in periodically well-populated regions. Compared to the single virtual mobile node-based algorithm, this algorithm has two advantages: (1) it tolerates the failure of one to all virtual mobile nodes; (2) as  $n$  grows, it correctly implements the neighbor detector with smaller values of  $\Delta_{predict}$ . Intuitively, this is because as  $n$  grows, the circular region is divided into more and consequently smaller subregions and each virtual mobile node spends less time to travel through its subregion. This feature makes the real-world deployment of the neighbor detector easier since with the existing prediction methods, loca-

tion predictions usually tend to become less accurate as  $\Delta_{predict}$  increases. We showed that the cost of our algorithm (in terms of communication) scales linearly with the number of virtual mobile nodes. We also proposed a set of optimizations that reduce the communication cost of the algorithm. We defined a set of explicit properties for the trajectory functions of the virtual mobile nodes to guarantee the coordination between them. We believe that our approach to define the trajectory functions of virtual mobile nodes can be used for implementing other virtual mobile node-based algorithms, such as virtual mobile node-based routing algorithms.

## 6.2 Future Work

The research conducted in this thesis can be extended in several directions:

**Energy efficient proximity-based durable broadcast.** We did not consider energy efficiency in the design and implementation of the spotcast variants. However, since mobile nodes tend to have very stringent energy limitations, the energy efficiency should also be considered as one of the requirements of proximity-based durable broadcast. There exist at least two approaches to address this problem: optimizing the implementations of the spotcast variants introduced in Chapter 2 or proposing new energy efficient algorithms for proximity-based durable broadcast.

**Design and deploy adaptive hello protocols for proximity-based neighbor detection in urban environments.** Based on the results of Chapter 3, adaptive hello protocols for proximity-based neighbor detection in urban environments can be designed and deployed. Such algorithms can adapt the transmission power and broadcast interval based on environment and application guarantees on effectiveness and efficiency. To deploy such algorithms on a smartphone, one can use lightweight sensing services such as the one introduced in [6], which can detect the indoor/outdoor environment in a fast, accurate, and efficient manner. Then, the results of the real deployment in terms of detection effectiveness and energy efficiency can be compared to the results of simulations in Chapter 3 and possible optimizations can be performed.

**Deploy the multiple virtual mobile nodes-based algorithm in a real network.** The neighbor detector algorithm presented in Chapter 5 can be deployed on

a real network. Once the neighbor detector algorithm is deployed on a real network, the theoretical results obtained in Chapter 5 can be compared with the results obtained from the real deployment. In fact, as shown by a quantitative analysis in Chapter 5, when  $n$  (i.e., the number of virtual mobile nodes) grows, the neighbor detector algorithm requires smaller values of  $\Delta_{predict}$  to correctly implement the time-limited neighbor detector service and at the same time its communication cost grows as  $\mathcal{O}(n)$ . Thus, it would be interesting to validate the theoretical results using the real deployment and determine up to which value of  $n$ , the utility of the algorithm outweighs its cost. In chapter 5, we have also proposed a set of optimizations for the algorithm. Thus, these optimizations can be applied on the deployment to determine whether they can effectively improve the performance of the algorithm.

**Proximity-based group formation.** One of the requirements of PBM applications that we did not consider in this thesis is *proximity-based group formation*. In some of existing PBM applications, a group is formed of people who are in proximity of each other for a period of time. An example of such applications is iGroups [1], a smartphone-based social networking application that enables people attending events, such as a concert, a trade show, a business meeting, a wedding or a rally, to form a group. Group formation is done anonymously and implicitly, without requiring users to send any message to each other. Once the event is over, people who were in the event are asked to confirm their membership to the group. If they do so, they receive the list and contact information of other members of the group. The current solution proposed in the literature [1] for solving this problem is to broadcast tokens using an ad hoc networking mode. All devices within transmission range of each other that are set in *Token Exchange mode* begin exchanging and storing tokens. After some time, each of the members upload their collected tokens to *the trusted service* (that can be a service such as Apple iCloud or any similar trusted third party that maintains a secure database). As this approach uses a hybrid architecture i.e., both infrastructure-based and ad hoc architectures, a possible research question to investigate is: *how to build a proximity-based group using a pure ad hoc architecture?*

**Privacy-preserving mechanisms for PBM applications.** One of the requirements of PBM applications that we did not consider in this thesis is *preserving privacy*. Thus, a possible research question to investigate in a future work is: *which privacy-preserving mechanisms should be considered for PBM applications?* This is a broad question which can be split into various subquestions. For instance, *what*

*type of privacy should be considered for PBM applications?* One possible answer is *location privacy*, i.e., users are concerned about the fact that their exact location at specific times can be obtained by others [2; 4; 5; 3]. Another subquestion can be: *is there a minimum location privacy that can be guaranteed by the abstractions?* and if the answer is yes, then, a possible question is: *are cryptographic and information security-preserving mechanisms can be used to achieve this minimum location privacy?*

## References

- [1] Apple's iGroups. <http://tinyurl.com/y9cwvmx>.
- [2] J. Freudiger, When whereabouts is no longer thereabouts : location privacy in wireless networks, EPFL Thesis, no 4928, 2011.
- [3] H. Nissenbaum, Privacy in context: technology, policy, and the integrity of social life, In *Stanford Law Books*, 2010.
- [4] D. Solove, A taxonomy of privacy, In *University of Pennsylvania Law Review*, pp. 154:477, 2005.
- [5] D. Solove, Understanding privacy, Harvard University Press, 2008.
- [6] P. Zhou, Y. Zheng, Z. Li, M. Li, and G. Shen, IODetector: a generic service for indoor outdoor detection, In *Proc. ACM SenSys'12*, pp 113–126, 2012.