UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CAMPOS: A CONTEXT-AWARE MODEL FOR POSITIONING IN OUTDOOR ENVIRONMENTS THAT SUPPORTS LOOSELY COUPLED MOBILE ACTIVITIES

TESIS PARA OPTAR AL GRADO DE DOCTOR EN
CIENCIAS MENCIÓN COMPUTACIÓN

DANIEL ANTONIO MORENO CÓRDOVA

PROFESOR GUÍA:
SERGIO OCHOA DELORENZI

MIEMBROS DE LA COMISIÓN:
NELSON BALOIAN TATARYAN
NANCY HITSCHFELD KAHLER
LEANDRO NAVARRO MOLDES

El trabajo presentado en esta tesis ha sido financiado por el Programa de Becas NIC Chile, y parcialmente por Fondecyt (Chile), Proyecto 1150252.

SANTIAGO DE CHILE
2017

Resumen

En escenarios ubicuos, conocer la posición de un dispositivo es imperativo para proveer al usuario de servicios personalizados basados en *location awareness*, un aspecto de diseño clave en la mayoría de las aplicaciones ubicuas que dependiente de las capacidades de los dispositivos para “sentir” cambios en su ambiente de trabajo. No existe una solución que aborde todos los tipos de posicionamiento, pues distintos tipos de aplicaciones requieren información de posicionamiento variada en términos de exactitud, precisión, complejidad, escalabilidad, y costo.

En escenarios ubicuos estándar, suele más de una estrategia de posicionamiento disponible, pero en general los dispositivos móviles no son capaces de determinar cuál es la más adecuada dado el contexto de trabajo del usuario. Además, este contexto está en constante cambio a medida que el usuario se mueve, perdiéndose conexiones a ciertos elementos del ambiente y ganándose otras. Aunque existen soluciones que abordan el posicionamiento en escenarios específicos de manera efectiva, hacerlo tomando en cuenta la mayoría de estos escenarios sigue siendo un problema abierto.

La propuesta presentada en esta tesis es un modelo de posicionamiento sensible al contexto (CAMPOS), que permite a dispositivos que realizan actividades débilmente acopladas en escenarios ad-hoc al aire libre, elegir estrategias de posicionamiento adecuadas a su contexto, basado en variables contextuales predefinidas. El modelo elabora un "catálogo" de estrategias disponibles y los puntos de referencia, usando las variables contextuales como entrada para un clasificador *RandomForest*, el cual determina un orden de “idoneidad” para las estrategias de posicionamiento, lo que permite acceder a estrategias ajustadas al contexto del usuario.

CAMPOS fue diseñado usando una metodología iterativa basada en casos de estudio. Primero, se realizó una revisión de literatura para determinar umbrales y valores promedio iniciales para las métricas y variables del modelo. Luego, se implementaron dos conjuntos de simulaciones; el primero para experimentar con distintos escenarios y configuraciones de dispositivos; y el segundo para evaluar el rendimiento del modelo. La batería de pruebas incluyó 27 plantillas de escenario, ejecutadas 15 veces para un total de 405 experimentos. Las variables observadas incluyen el efecto de variar la cantidad de *beacons* (dispositivos con capacidad de posicionamiento), la cantidad total de dispositivos, y el rango de comunicación. Todos los experimentos presentados en este trabajo se realizaron utilizando el ns-3, un simulador de redes de eventos discretos orientado a la investigación.

El aporte de CAMPOS reside en que no es una nueva propuesta de estrategia de posicionamiento, ni busca mejorar el estado del arte en términos de precisión. En vez de ello, proporciona a los dispositivos de una red los medios para censar su entorno y determinar qué estrategia de posicionamiento es más adecuada para su contexto. Además, dado que CAMPOS es independiente del proceso formal de posicionamiento, si apareciesen nuevas estrategias de posicionamiento en el futuro, éstas podrían añadirse a CAMPOS con relativa facilidad, permitiendo que los dispositivos potencialmente tengan acceso a dichas estrategias a través del modelo.

Abstract

In mobile, ubiquitous scenarios, knowing a device's position is mandatory in order to provide *location awareness* and personalized services. This design aspect is key for most ubiquitous applications, and strongly dependent on the device's capabilities to sense changes in its environment. However, there is no one-size-fits-all solution to provide positioning to devices. Different types of mobile applications may require positioning information that varies in terms of accuracy, precision, complexity, scalability, cost, and deployment effort.

Thus, in a standard ubiquitous scenario, one or more positioning methods could available, but the devices usually have no clear way of determining which is better suited to the user's situation in relation to its environment, i.e., its work context. Moreover, this context is constantly changing as the user moves; while some connections to other devices and appliances are lost, new ones also become reachable. Although there are good solutions for addressing positioning in particular scenarios, doing so while considering most of these scenarios is still an open problem.

This thesis work proposes a *context-aware positioning model* (CAMPOS) that addresses this problem by allowing devices performing loosely-coupled activities in ad hoc outdoor scenarios, to pick a suitable positioning strategy based on a set of predefined contextual variables (e.g., scenario embedded instrumentation, neighboring devices, etc.). The model assembles a "catalog" of available positioning methods and reference points, and uses the contextual variables as features for a Random Forest classifier. The classifier ranks all positioning strategies from the catalog in order of "suitability" for the device's given context, allowing the use of strategies tailored to the user's context rather than the default ones, and allowing devices without access to positioning strategies to perform trilateration positioning on the go.

CAMPOS was designed using an iterative methodology based on case studies. First, a review of literature was performed to determine thresholds and average values for most of the metrics and model variables. Then, a set of simulations was implemented to experiment with different scenario and device configurations, and another to evaluate the performance of the model. The test battery included 27 experiment templates, each of which was executed 15 times, for a total of 405 experiments. During each simulation, we observed the effect of the variation of the amount of beacons, total population, and communication range. All experiments presented in this work were performed using the ns-3, a discrete-event networking simulator oriented towards research.

The novelty of this thesis work lies in the fact that it is not focused on proposing a new positioning method, or improving the state-of-the-art in terms of accuracy. Instead, it provides all devices with means to assess their environment and determine which positioning strategy is more suitable for their given contextual situation. In addition, new positioning techniques can be included into CAMPOS with relative ease, because it is independent from the actual positioning. In this case, the new positioning strategy could be added to the model's pool of recognized positioning strategies, and thus become an available option for the participating devices.

Publications

- Moreno D., Ochoa SF. **Understanding the Resource Positioning Methods that Support Mobile Collaboration**, 2016 IEEE International Conference on Systems, Man and Cybernetics (SMC2016), pp. 3676-3682. October 2016, Budapest, Hungary.

- Moreno D., Ochoa SF., Meseguer R. **Providing ubiquitous positioning in outdoor environments**, IEEE International Conference on Systems, Man and Cybernetics (SMC2015), pp. 1285-1290.Hong Kong, China, October 2015.

- Moreno D., Ochoa SF., Meseguer R. **A context-aware model to provide positioning in disaster relief scenarios**, Sensors 15(10), pp. 25176-25207. Sep. 2015.

- Moreno D., Ochoa SF., Meseguer R. **Understanding confidence of positioning measurements in collaborative outdoor environments**, IEEE International Conference on Systems, Man and Cybernetics (SMC2014), pp. 1141-1146. San Diego, CA, USA, October 2014.

- Moreno D. **Context-aware Positioning Model to Support Ubiquitous Applications**, Readings of the SADUE13 International PhD Workshop 2013, Santiago, Chile.

- Moreno, D., Ochoa, S. F., Santos, R., Meseguer, R. **Geo-localized messages irradiation using smartphones: An energy consumption analysis**, IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2013), pp. 681-685. Whistler, BC, Canada, June 2013.

- Moreno D. **Survey on Resource Positioning**, Technical Report N° TR/DCC-2013-1. Computer Science Department, University of Chile, Santiago, Chile, 2013. Available at: http://swp.dcc.uchile.cl/TR/2013/TR_DCC-20130102-001.pdf.

Acknowledgements

To my best friend and wife Teresa, without whom I would never have finished my PhD. It was a long and difficult road, but we reached the end.

To my little cheerleaders, Marco and Andrea, who were as much a distraction as an inspiration to finish writing the thesis. Go to sleep already!

To my family. My mom Carmen and my sister Diana, who have been waiting for news of my graduation for years. My dad Antonio and my grandparents Carmen and JJ, who saw me leave my hometown one day, and whom I will never get to see again. My grandfather Jorge, and too many names to put in such a short space. Thanks to all of them for their support during these long years.

To my extended family, my parents in law Elio and Violeta, who always treated me like their own son. To Milagros, who was part of our family in Santiago for a while, and for helping us raise Marco. To Diego, for giving me enough funny stories to last for a lifetime. The “woOoOolf!!!” one never get old.

To Iván and Analí, for giving us a warm welcome and accompanying us through most of our stay in Santiago. Carlos, for being there for us and lending us help when we direly needed it. To the Peruvian colony, who provided solace from the darkness of doctorate studies.

To my advisor, Sergio, who guided me through the process of becoming a Phd. Sandra and Angélica, who guided us through the sea of bureaucracy in which we would otherwise become lost. And last but by no means least, to everyone in the DCC. It was great sharing the building with all of you during last four years. David, Luis, Francisco, Maira, Pablo, Ismael, and so many others.

Thanks for all your encouragement!

Honorary mention for Nic Chile, Fondecyt, and Sergio for providing funding and assistance for my research.

Contents

Index of Tables

Index of Illustrations

Index of Algorithms

Chapter 1: Introduction

The global population of Internet users has undergone an unforeseen growth in the past fifteen years, going from almost 400 million users in the year 2000 (roughly 6.5% of the world's total population) to over 3,200 million in 2015 (more than 43% of the population). Since the arrival of Internet-capable mobile devices, such as smartphones or tablet PCs, these users have preferred these devices due to their portability, a trend that has intensified over the past few years (Fig.1-1). Several studies [90, 117, 126] show that people tends to rely more and more on their mobile devices as support for their daily activities (Fig. 1-2). This support ranges from using social media to interact with other people, to on-demand real-time surveillance of their homes [127, 170], to heart-rate monitoring devices [58, 66] for people with heart diseases. Moreover, it is expected that this tendency only becomes stronger with the proliferation of smart devices and smart spaces.

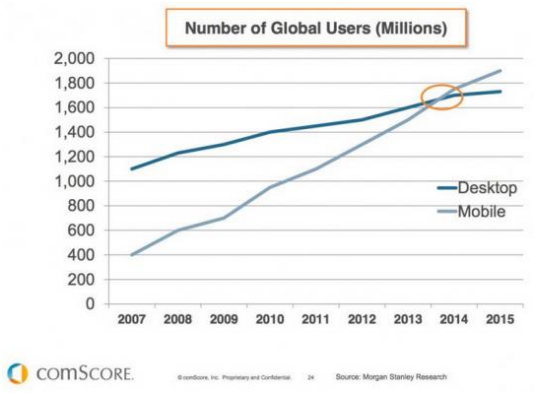


Figure 1-1. Internet Penetration Prediction [116]

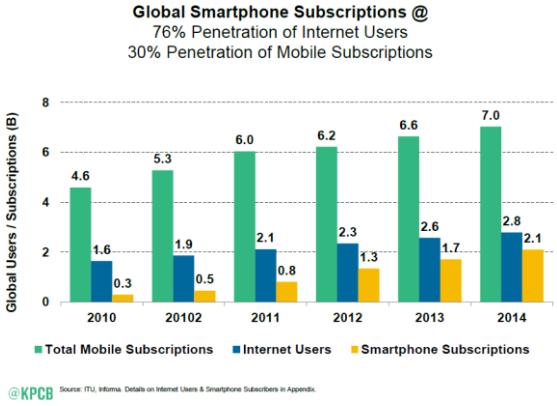


Figure 1-2. Use of digital media in USA [102]

The always connected paradigm [128] increases the need of users to carry Internet connectivity with them at all times, in order to interact with other people and with their environment in a simple and ubiquitous manner. The Internet-of-Things (IoT), wearable and smart devices, and smart environments, are a reality that must be considered in the design of the next generation of mobile and ubiquitous computing services and applications.

Recent studies predict that an important growth in the use of wearable devices will occur over the course of the next five years [29, 75], which will change the current computing scenario (Fig. 1-3). The new computing scenario will become more and more heterogeneous in terms of device specializations, computing power, and portability. Device autonomy and interoperability will become fundamental aspects to be considered

in these solutions, since replicating services in multiple devices (e.g., positioning capability or data capture) does not make sense and adds to the size, weight, and cost of these devices, particularly to the smallest ones (such as wearables).

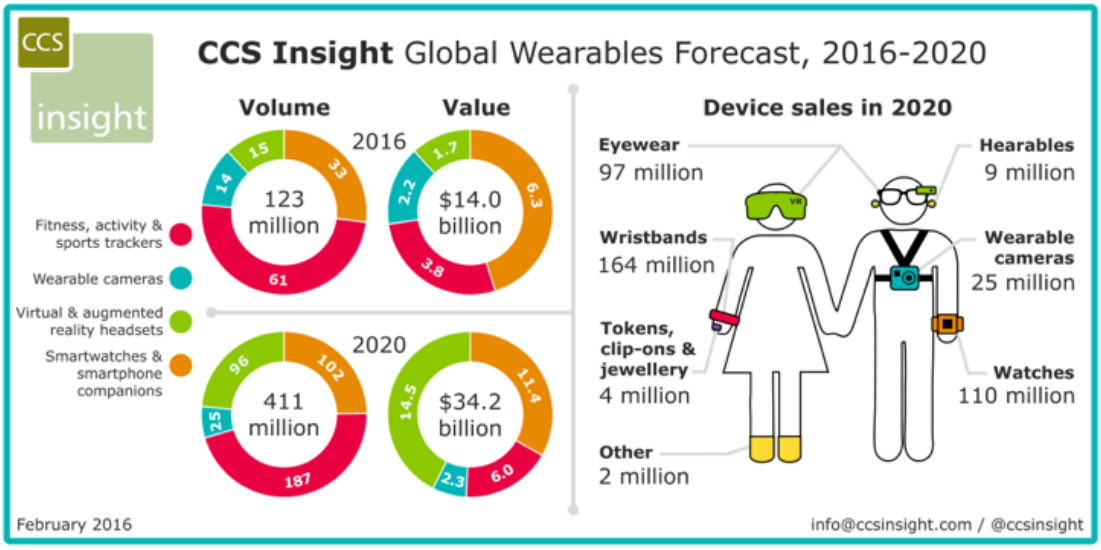


Figure 1-3. CCS Insight's Global Wearable Devices Forecast 2016-2020 [29]

This new computing approach, promoted by the IoT paradigm, conceives the capabilities of mobile (and wearable) computing devices as services that can be provided and consumed by other devices. Thus, a device without sensing or positioning capability could count on such information through collaboration with neighboring or remote devices. This avoids the limitation that every device participating in mobile collaboration processes can only do so if they are able to provide all the services required to be part of such a process.

The advances in wireless communication and mobile computing have opened several opportunities to use technology to support the people activity, not only individual work, but also mobile collaboration. The following section introduces the concept of mobile collaboration and its main application domains.

1.1. Mobile Collaboration

A Mobile Collaborative System enables the members of a team to communicate and coordinate their activities, allowing them to achieve a common goal, usually in a loosely-coupled manner, by using a combination of mobile and stationary technologies [12]. A mobile collaboration system generally includes (1) a wireless resource controller for receiving and/or transmitting data through a wireless medium, (2) a management module for receiving and/or transmitting data and controlling the operation of the mobile collaboration and communication system, (3) a policy database for storing a plurality of levels of authorization for network access, and (4) a communication module, for creating

and managing collaboration and coordination, and relaying and controlling the data traffic to and from the electronic devices [169]

Mobile computing supported by wireless technologies is widely used in several mobile collaboration scenarios; e.g., hospital work [8, 153], logistics [123, 156], security [27, 31], transportation [86, 165], tourism [42, 96], and emergency responses [79, 121]. The way in which the users participate in the collaboration defines functional and non-functional requirements, as well as restrictions to the software applications they use. For instance, they could conduct participatory sensing, crowd-sensing, opportunistic sensing, and they can do it in a loosely-coupled or tightly-coupled manner. These configurations enable particular functionality and pose additional requirements to applications that support these activities.

Particularly, participatory sensing [24, 133] involves the *voluntary* cooperation among users in order to gather, share and eventually process information, typically involving their local context [11]. These activities require the explicit participation of the users, although the interaction among them is usually indirect. Similarly, crowdsensing [55, 62] also requires user intervention to provide sensing information. However, it reuses data explicitly entered by the users through Internet services, such as social networks. Typically, these sensing approaches use infrastructure-based communication that allows mobile users to access centralized shared information repositories; therefore, these collaboration approaches can be used if the devices and the environment count on such a communication infrastructure.

Opportunistic sensing [108, 133] uses a different strategy to gather and share information. This collaboration approach provides a method to capture context information automatically from sensors (e.g., those embedded in a smartphone), and eventually share it with other neighboring or remote devices. In this case, the user is not directly involved in the data collection process, and the information is sensed unobtrusively. Services considered in pervasive computing or unattended mobile collaboration can take advantage of opportunistic sensing.

On the other hand, the direct collaboration among mobile users can be classified as loosely or tightly-coupled. The former is the most frequent, since it does not require a stable or permanent communication link among the participants. In loosely-coupled activities, the mobile users typically work autonomously most of the time, while sporadically performing on-demand collaboration processes; i.e., tightly-coupled activities (Fig. 1-4) [35, 68, 129]. After engaging in collaboration, the users return to autonomous work. This collaboration style is usually supported by ad hoc and opportunistic interactions.

Contrarily, tightly-coupled work involves the use of synchronous mobile applications, which is not recommended for most collaboration scenarios due to the systems

dependency on the availability of a stable communication link among the participants (and eventually, remote resources). This collaboration style usually involves the use of centralized components, which must be available during user interaction.

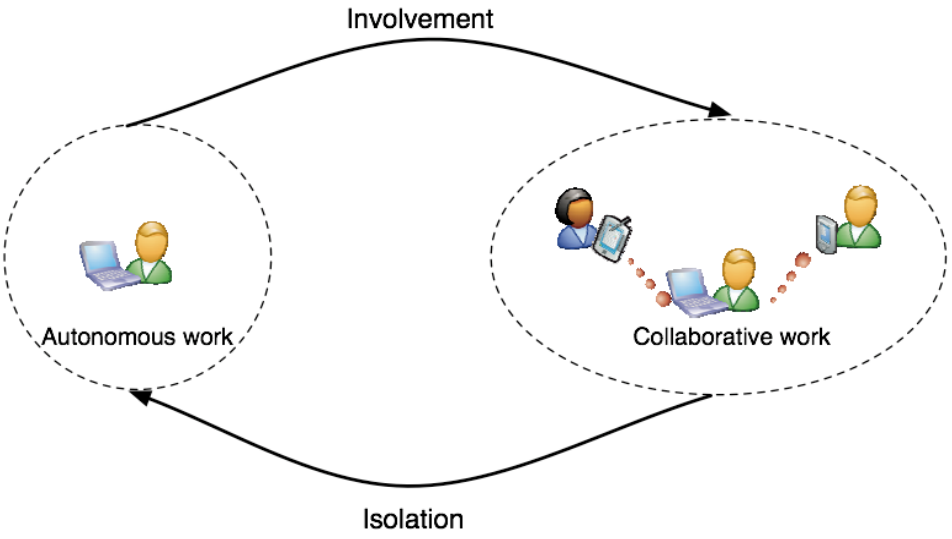


Figure 1-4. Work in mobile collaboration [68]

Today, a large number of services provided or consumed by mobile users are highly dependent on the users' context, particularly their location [34]. Therefore, there is an important demand for accurate indoor and outdoor positioning strategies to support mobile collaboration [60, 69, 125]. By means of a positioning technique, a mobile device can either gather the information about its position (e.g., context information) or it can be localized from elsewhere.

This strongly emerging interest in positioning is driven by several factors. At first, the great success of wireless systems is essentially explained by the mobility they enable, which is coupled with uncertainty. However, this uncertainty is often not desired in most of the application domains, and the only means to efficiently overcome this limitation is to know the position of our assets (including people, and stationary and mobile resources). Data security and integrity (ensuring that the data is safe from interlopers, and that it has not been tampered with, respectively) also benefit strongly from local positioning, because by using information on the origin, propagation path, and destination of the data, the participants have means to detect interlopers and disregard messages that come from outside of their intended work area. Last but not least, the data capacity of wireless networks is inherently limited, so an intelligent context-dependent information transfer is needed [163]. One essential context service to address this issue is the positioning of the participating resources.

Positioning can be roughly divided in two categories, based on the environment in which they work best: outdoor positioning, and indoor positioning. In outdoor environments, the GPS (a satellite-based positioning system), is currently the most widely

used, offering maximum coverage with relatively little effort [70]. However, GPS cannot be deployed for indoor use, because it requires having line-of-sight (LOS) transmission between receivers and satellites. Indoor scenarios are inherently more complex that outdoors, due to obstacles (e.g., walls, equipment, etc.) [87].

Some authors believe that the positioning problem for outdoor environments has a concrete solution in the form of the Global Positioning System [59]. Nevertheless, such a solution requires that each participating device has GPS, which is not always feasible. The diversification of mobile computing devices (including clothes) and the integration old and new technology to address particular problems, create the need of new solutions, able to deal with the heterogeneity of devices.

In addition, mobile collaborative applications require different types of positioning information, or varying degrees of accuracy. For instance, the most widely used coordinate systems are degree/minutes/seconds (DMS), degree decimal minutes, and universal transverse mercator (UTM) [69]. In addition, the accuracy requirement varies greatly depending on the type of consumer application, ranging from Kilometers to up to few centimeters, depending on the requesting application.

Given the previously stated limitations, this work is particularly focused on outdoor positioning, considering heterogeneous devices participating in mobile collaboration process, even if they do not have positioning capability. Although this research also considers supporting mobile collaboration activities, such as crowdsourcing, participatory sensing, and opportunistic sensing, the largest portion of the research effort is focused on supporting loosely-coupled activities in ad hoc scenarios.

1.2. Problem Statement

In mobile computing environments, users might need to interact with both their devices and their surroundings, because everyday more and more technology components are embedded in the physical scenario; i.e., in their physical context. Figure 1-5 portrays an example of such a scenario, in which users pursue different activities while interacting with their surroundings through their portable devices. Either taking a walk around the city, visiting landmarks, or travelling by vehicle, users have access to several services available in the environment, most of which benefit from having access to positioning services.

Ubiquitous systems, which are a particular flavor of collaborative systems, typically involve several autonomous computational devices of modern life, ranging from consumer electronics to smartphones. Ideally, users do not notice the presence of these devices in their physical environment, but they still benefit from the services provided by these computing artifacts [97].

In order to provide adequate services to address the users' needs, most context-

aware applications require knowing the position of the user or other resources (e.g., access points, public video-cameras, or other users). If a device is unable to obtain its position or if the positioning error is too high, any service provided by the supporting application might not work properly, jeopardizing the collaboration process. Moreover, attempting to reduce the positioning error of a given estimation could lead to several additional positioning attempts, which could yield similar results at the cost of additional energy consumption.

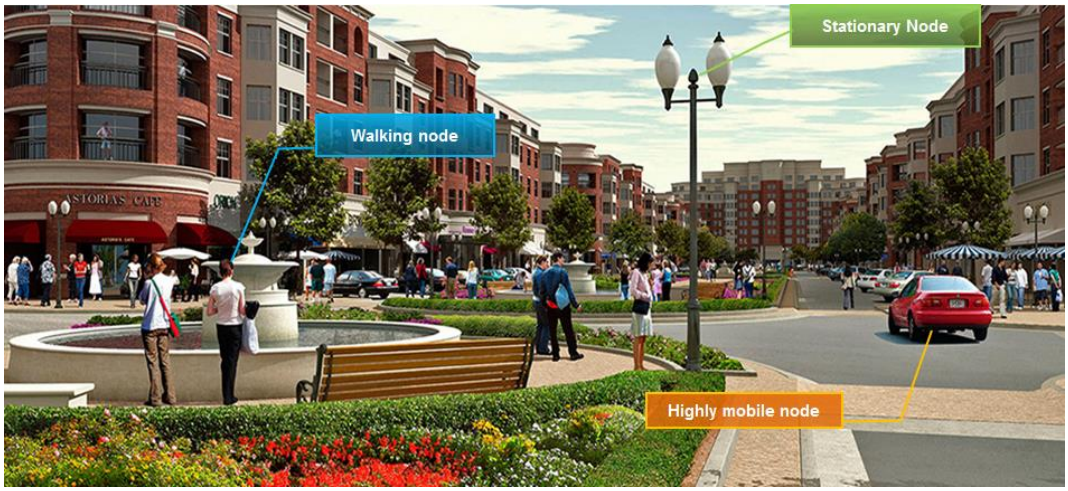


Figure 1-5. Example of a collaborative outdoor environment

In some cases, the participating devices that have access to several types of positioning strategies generally commit to a single one (mainly GPS), even if using another strategy could provide better results in terms of response time or accuracy. Therefore, the devices involved in mobile computing activities need to determine their best positioning alternative, even under unfavorable or limited conditions. Moreover, not all of the participating devices possess peripherals that enable to perform positioning on their own, and not all of them have these peripherals enabled at all times. Thus, they would be not able to take advantage of services embedded in their surroundings, unless they count on external support for performing positioning.

Therefore, we can safely assume that there are instances where a mobile device cannot perform positioning when required, even if one or more positioning systems or strategies are available in the environment. Examples of such situations include:

- The target device has no positioning capabilities whatsoever.
- The target device has positioning capabilities, but is unable to access positioning strategies available on the scenario, due to issues in the local or remote node (i.e., obstructed line of sight towards GPS satellites, server overload, network flooding).
- The target device manages to estimate its position, but the estimation error is too large to be useful.

Additionally, *energy* is usually a valuable and scarce asset for mobile devices, which

is not always readily available on mobile environments; therefore, it must be taken into account both, when selecting the positioning strategy to be used, and also when performing the positioning itself. Allowing positioning services to spend large amounts of energy by focusing on maximizing accuracy does not make sense in all scenarios, since it reduces the time period in which the collaborative system is available. This leads us to postulate the following research questions:

RQ 1: How to enable all or most of devices in a given outdoor scenario to estimate their positions, even if they have limited or no positioning capabilities of their own?

RQ 2: How to achieve the former without increasing considerably the energy consumption of both individual devices and the entire network?

1.3. Work Hypotheses

Considering the stated problem and the research questions presented in the previous section, we have defined the following work hypotheses:

H1: Devices in outdoor environments could use their peripherals and communications capabilities to sense their environment, effectively assembling a working-context that includes all elements relevant to the positioning process.

H2: Devices with positioning capabilities could make use of their contextual information to determine which positioning strategy is better suited given their current context, and taking into account factors such as accuracy, response time and energy consumption.

H3: Devices with no self-positioning capabilities that are part of a mobile network could collaborate with their close neighbors, sharing positioning information (if available) and using it to obtain a rough estimate of their position.

1.4. Thesis Goals

The main objective of this thesis work is to propose a *context-aware model for positioning* (CAMPOS) that allows devices in outdoor environments to obtain their positions, regardless of their capabilities. The model must allow most or all of the participant devices to perform outdoor positioning, includes devices that do not have proper positioning peripherals (i.e., no GPS receiver, no RFID reader). In addition, the model must support the sharing of positioning information among participants. In terms of energy consumption, the model must attempt to maintain the overall consumption of the positioning process within levels similar to those of self-positioning strategies (namely, GPS).

A device using the model proposed in this thesis should be able to:

- **Sense changes in its context, and adapt its behavior accordingly.** Devices

using the model should have the capability to sense their environment and determine whether to perform positioning or not (and how), based on contextual information.

- **Collaborate with neighboring devices by sharing positioning information.** The participant devices should be able to request and/or relay positioning information to their peers, potentially enabling all of them to estimate their positions.
- **Avoid unnecessary energy consumption.** Devices utilizing the model can decide not to attempt to perform positioning if conditions are not suitable, thus saving energy. The average energy consumption of devices utilizing the model to perform positioning should not exceed that of devices that do not use the model.
- **Require no additional instrumentation.** Devices should be able to work with what is available in the environment, without the need of additional instrumentation or equipment.

It is important to note that we are not attempting to maximize the accuracy of the positioning estimates, tracking the users, or relaying messages through the network (aside from positioning information). Our focus is on providing “tailored” collaborative positioning based on the context of a device, in outdoor environments.

1.5. Applications Scenarios

In this section, we illustrate three possible application scenarios of this proposal, as a way to help the reader in understanding the application domains of the model.

1.5.1. Disaster Relief Efforts

Several countries in South America and the Asia-Pacific region are frequently affected by natural disasters, such as earthquakes, tsunamis, landslides, and volcano eruptions. When these extreme events hit urban areas, a first response process is triggered and several emergency response companies are deployed in the affected area; e.g., firefighters, police, government agencies medical personnel, and eventually military units [120]. Every organization is in charge of specific activities that range from performing search and rescue of victims to secure particular working perimeters.

Typically, electric energy is not available in the affected area, because the power distribution network is usually damaged or turned off for security reasons. Similarly, the infrastructure-based communication network is frequently damaged or collapsed; therefore the communication is mediated mainly by very high / ultra-high frequency (VHF/UHF) radio systems [121]. Figure 1-6 provides an example of an emergency scenario, showing the rescue efforts of firemen, policemen, and military personnel while aiding in the rescue of trapped people in a collapsed building after the February 2016 earthquake near Pingtung City, Taiwan.

In such a scenario, the most critical response activities should be performed during the first 72 hours after the event (also known as the golden relief time), since after such a period the probability of rescuing people alive in the affected area is quite low [43, 119]. These activities include search and rescue of victims, providing first-aid to injured people, diagnosing the stability of physical infrastructure, establishing evacuation and re-entry routes, and securing the working area. All these activities are interconnected and require coordination among the response organizations. Coordination has a direct impact on the effectiveness of the initial response to the emergency.

Command posts deployed in the field try to coordinate the local activities, but they usually fail due to the unavailability of digital communication support and geo-referenced information that would allow them to know the status of the first response process [99, 148]. Therefore, improvisation becomes the common denominator in those scenarios, which reduces the effectiveness of these processes [16, 106].

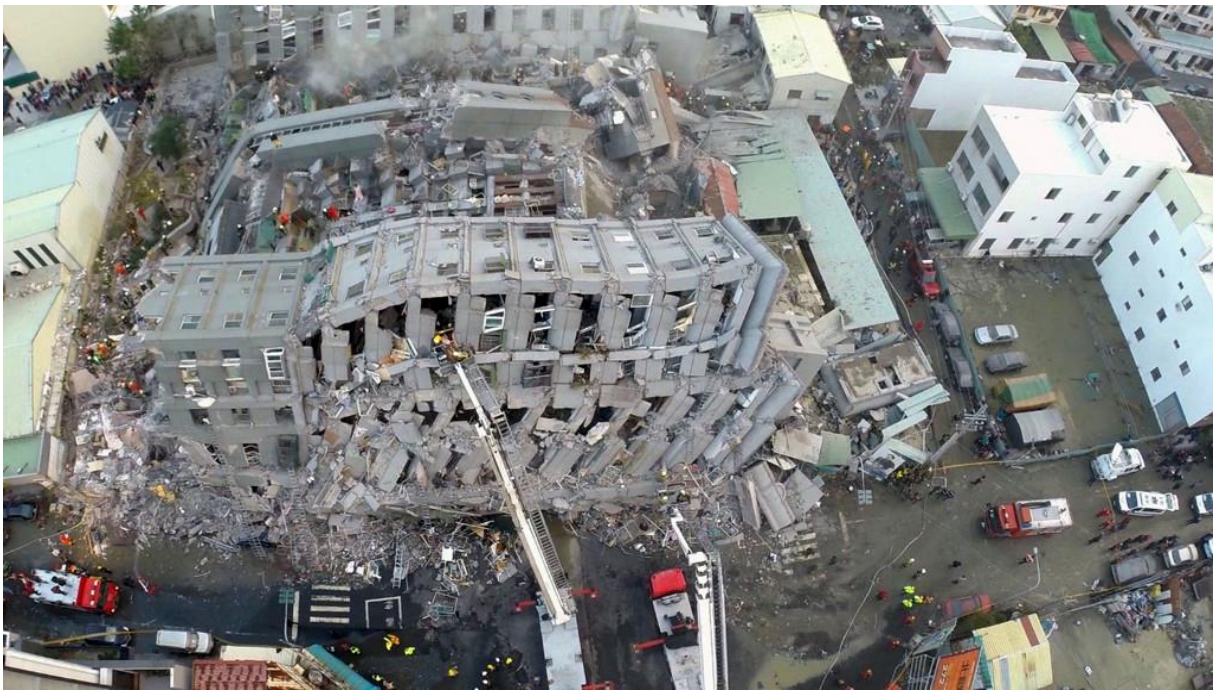


Figure 1-6. Fallen building after an earthquake, Taiwan

Most research work done to address this coordination problem is focused on providing digital communication in the field, in order to support interactions among first responders and positioning of the available resources (including rescuers) [15, 50, 53, 99, 135]. This process involves manned and unmanned vehicles, mobile workers, mobile communication brokers (mules), mobile and stationary sensors, and intermediary devices (typically, information temporal repositories). Many of these devices do not possess positioning capabilities, although most of them require some degree of positioning. Therefore, a positioning method that considers such a diversity would be very welcome to support the coordination of these activities, and therefore to increase the effectiveness of

the first response process.

This is particularly relevant for response organizations belonging to developing countries (e.g., all of Latin America), where the most important one (firefighters) is composed almost exclusively of volunteers, financed mainly by the community with little economic support of the government. These organizations generally use several outdated mobile computing devices donated to them by the community (e.g., cellular phones, tablet and laptops), which are installed in the fire trucks to store and exchange digital information with first responders, and with mobile and stationary resources. Although the police, medical units, and military forces generally have more technology to support their activities, not all of their resources have positioning capabilities, or can position resources in the affected area; e.g., under roof, or in partially collapsed buildings, where the access to satellite signals are limited.

Moreover, all these organizations would benefit from knowing the position of some of the firefighters' resources (e.g., rescue teams travelling through the evacuation route towards the first aids points) while performing their activities. Therefore, providing firefighters with positioning also allows more coordination during rescue activities, and improves the performance of the response activities of all involved organizations.

A context-aware positioning model that takes advantage of computing devices with positioning capability (e.g., those embedding GPS), in order to provide positioning to those who are not able to do it by themselves, would represent an important contribution in this scenario.

1.5.2. Tourism/Leisure

The leisure and tourism sector is extremely heterogeneous [146] in terms of the activities that the users perform during their off-time, and also how these activities can be aided by ubiquitous applications. For example, some people use their free time to travel abroad, visiting new countries, exotic locations, and the like; while others enjoy visiting meeting places from their own or from neighboring cities, such as malls, amusement parks, or beaches. In these scenarios, several venues of application for positioning become available, including social networking, point of interest (POI) research for visiting and meeting, traffic information, urban navigation, automated tourism services, theft prevention, and personal tracking [172].

Most mobile users take advantage of geo-tagging to add value to their social network publications, serving as a means to "prove" that they actually traveled to a certain location, or that they attended a certain event. For instance, a traveler might post a Facebook status with a picture of a sunset, adding a geo-tag for the Caribbean beach he or she is currently visiting. Another user could post a "selfie" on Instagram, with a concert stage as a background with a band playing, with a geo-tag that shows his location within the concert's venue.

On a different venue, automated tourism services offer context-aware or location-aware travel information to tourists, including guides to certain parts of supported cities, restaurants, history of the place, and POI guides. Examples of automated tourism systems include GUIDE [32], COMPASS [161], CRUMPET [147], and SightSeeing4U [144]. The main goal of these systems is enabling users to get the most out of their visits to supported locations, providing contextual information related to their current position within the complex. For instance, Figure 1-7 shows tourists visiting the Louvre museum. These users can access information related to Etruscan and Roman history and anecdotes while visiting the Denon Pavilion, with detailed information updating on their smart device as they move through the exhibits.

Figure 1-7. Denon Pavilion of the Louvre Museum, Paris, France

Positioning applied to traffic information and urban navigation provides tourists and locals alike with information related to the status of roads and routes through cities. In the case of traffic information, knowing their position allows users to geo-tag events occurring in roads, such as accidents, blockades, strike marches, or potential traffic jams, enabling other users to visualize these events and avoid them.

Urban navigation services provide users with the location of nearby POIs to users, including restaurants, fuel stations, parking lots, plazas, malls, historical buildings, etc. In addition, routes can be traced to go from one point in the map to another, making a differentiation between the types users, which can be vehicles (restricted to roads and highways), or pedestrians. Some urban navigation systems even allow tracing water

routes, such as the Marine Navigation application [101] from the Netherlands.

In terms of personal security, theft prevention systems provide people (including tourists) the ability to track the location of assets, such as vehicles or devices. These systems work either reactively, or passively; users can visualize the current location of their marked assets, or set alarms to fire when the location of said assets changes without permission. This way, users are able not only to determine whether something of their property has been stolen, but also where such a product is located.

Similarly, personal tracking systems allow users to determine the position of tagged individuals, such as children, pets, and even mentally ill persons (e.g., a Parkinson disease patient, or a person with autism spectrum disorder), so that they do not get lost while visiting places unfamiliar to them. These systems also work reactively or passively, by sending alerts to the caretaker, or showing the target's location on demand.

If the proposed context-aware positioning model was to be deployed in these scenarios, the need for a centralized location strategy would become less imperative, and it could be delegated to other positioning methods that offer similar results.

1.5.3. Logistics

Logistics can be defined as a business planning framework for the management of material, service, information and capital flows, including the increasingly complex information, communication and control systems required in today's business environment [95]. In logistics, the supply chain management of assets is a key activity that constitutes, on average, the highest portion of the total logistics-related costs [56]. An area with significant potential for research is dynamic planning, specifically dynamic re-scheduling and re-routing of vehicles, which of late has become more relevant due to the emergence of technologies enabling real-time, high-bandwidth information exchange between fleet vehicles and/or between a vehicle and its headquarters.

Supply chain management processes can be classified in two major categories: planning and execution. While supply chain planning deals with processes related to determining material requirements, planning for production and distribution, etc., supply chain execution focuses on the actual implementation of the supply chain plan, comprising processes, such as production and stock control, warehouse management, transportation, and delivery [56].

The transportation of merchandise from one place to another, i.e., "cargo", can occur through a variety of transportation means and it is organized in different shipment categories. Unit loads of similar types are usually assembled into higher standardized units, such as ISO containers, swap bodies, or semi-trailers; which can be transported by train, road vehicles, boats, airplanes, couriers, freight forwarders, and multi-modal transport operators. When moving cargo, typical constraints are maximum weight and

volume, which must be taken into account when planning transportation.

Handling systems, which deals with the load, unload, transport, storage, and retrieval process of assets within a certain complex, includes handlers (e.g., trans-pallet, counterweight, or stacker handlers) and storage systems (i.e., pile stocking, cell racks cantilever racks and gravity racks. Assets are picked either manually or automatically, sorted, and carried to their destination, mostly either storage or transport. Manual picking can be man-to-goods, i.e., operator using a cart or conveyor belt; or goods-to-man, i.e., the operator benefiting from the presence of a mini-load carousel or an AVSS (Automatic Vertical Storage System). Automatic picking is done either with dispensers or depalletizing robots. Sorting can be done manually through carts or conveyor belts, or automatically through sorters.

Figure 1-8. Container storage and transport logistics in DuisPort, Duisburg

To help grasp the actual scope of logistics and stress the need for location services, Figure 1-8 portrays Duisport, the largest inland port in the world. Each year, more than 40 million tons of various goods are handled with more than 20,000 ships calling at the port. The public harbor facilities stretch across an area of 7.4 Km^2, with 21 docks covering an area of 1.8 $Km^{2,}$ and 40 Km of wharf. A number of companies run their own private docks, and based on 2010 estimations, 114 million tons of goods are handled in Duisburg yearly.

An application of positioning to supply management is autonomous control. The idea of autonomous control is to develop decentralized, non-hierarchical planning and controlling methods for the automation of processes. Logistic objects are defined as

material items (e.g., vehicles, storage areas) or immaterial items (e.g., customer orders). These items can interact with other logistic objects within the system, and autonomous logistic objects are able to act independently according to their own objectives and navigate through the logistic network on their own [14], using their position to trace routes and determine proximity to target objects and tasks.

The benefit of applying positioning to cargo is more direct in the sense that the user will know the position of his/her assets on demand, with the possibility of visualizing not only its current location, but potentially also the route and arrival estimations if provided by the charter company in charge of transportation. The benefit is similar for the handling of assets, which would benefit from positioning by letting owners, transporters, and all other interested parties the location of the assets in question.

1.6. Thesis Contribution

This thesis work presents a context-aware positioning model (named CAMPOS) that enables all the devices (or most of them) in an outdoor scenario, to estimate their position based on information they gather from their direct context (i.e., their environment). This way, participating devices would be able to determine the best course of action attempting to perform positioning, doing so in a transparent manner (i.e., the user does not need to intervene in the positioning process), based on the information they can sense from their surroundings.

Users of the model would be able to obtain their positions under conditions they would normally be unable to, or to save energy by letting the model choose less energy expensive positioning strategies than the *de facto* strategy. This would grant devices that have limited or no sensing or positioning capabilities, to access their position via collaboration, as well as allowing devices that do have positioning capabilities to determine which available positioning strategy provides greater benefits in the long run.

CAMPOS would be especially suitable in the event of natural or intentional disasters, such as earthquakes, landslides, or bomb attacks, in which the supporting communication and power infrastructures are usually taken down due to severe damage, or for safety reasons. The model could help the relief efforts of volunteers, firemen, and military personnel by providing context-aware energy-wise positioning during their search-and-rescue and sweeping operations. However, the model is not limited to this type of scenario, and it could also be used in several others, including tourism [146, 147, 149], logistics [14, 47, 56], surveillance [105, 159], and analysis of behavioral models [2, 3], to name a few. Figure 1-9 shows a brief overview of how the model works. First, the environment is sensed. Then, the sensed context is evaluated. Finally, based on the previous evaluation, the most suitable positioning strategy available is chosen.

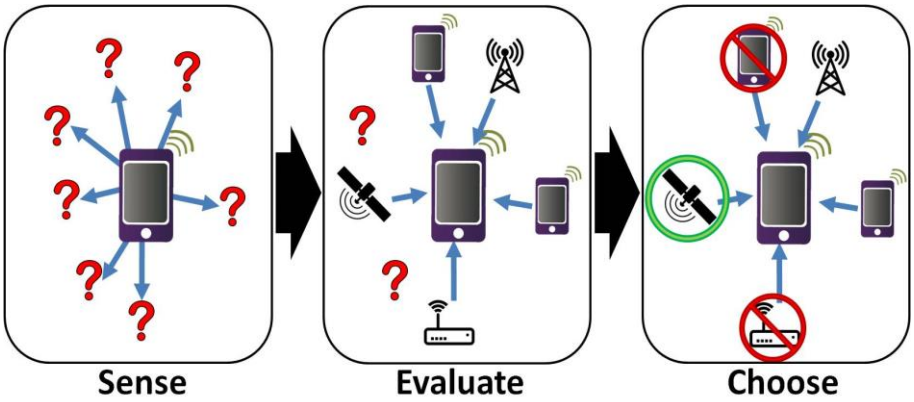


Figure 1-9. Overview of the proposed method

Most of the applied proposals presented in the related work are focused on increasing the accuracy of the positioning estimates, either through the use of more than one strategy at the time or by taking advantage of collaboration with neighboring nodes, paying little or no mind to other devices that do not possess the peripherals required to perform positioning. The proposed positioning model, on the other hand, focuses on providing these devices with the capacity to estimate their positions, and providing both devices with and without positioning capabilities with the most suitable available strategy given their context.

1.7. Limitations of the Model

This section details the limitations of the proposed context-aware positioning model. Each limitation is discussed pointing out its impact on our research, and also possible ways to overcome them are indicated in future work.

1.7.1. Real-world validation of the model

Similar to most thesis works in this area, the experimentation process was performed using simulations. Due to the collaborative nature of the model, it was almost impossible to conduct real-world validations involving several of experimentation settings. Although this aspect could be an issue, the research presented in this document was implemented on ns-3 network simulator [67, 154], using outdoor communication models implemented in such a simulator, which are broadly accepted by the research community of the area and also allow to reproduce the experiments to make in-detail analysis. Moreover, the mobility and setting models used in the simulations are well accepted by the research community as valid counterparts to real-world scenarios, and they have already been extensively used in networking research [54, 77, 84, 132, 138, 150, 155].

For more technical depth on the subject of ns-3 and network simulations, please refer to Chapter 5.

1.7.2. Indoor Positioning

The positioning model has been tailored to work based on the conditions present in outdoor scenarios, and it does not specifically address aspects involved in indoor positioning, due to the inherent difference between these types of environments. Specifically, indoor environments present several types of obstacles, including walls, the building geometry and construction material, electronic equipment, and even human beings and their devices. In addition, interference and noise sources from wired and wireless networks also influence the signals from satellites and exterior reference points, such as antennas and access points.

The presence of these elements influences the propagation of electromagnetic waves, leading to multipath effects, non-Line-of-Sight, scattering and other false readings, which in turn affects the positioning estimations. Given that CAMPOS has been designed with outdoor environments in mind, it is not possible for our current version of the model to support indoor positioning in an accurate way.

However, devices on indoor environments could still benefit and/or contribute in the collaboration process of the model, if the conditions for positioning are suitable (e.g., they have unobstructed line-of-sight to an outdoor participant, or are close to a window). Moreover, given that the model is detached from the actual positioning strategies, future work could be focused on including complete support for indoor environments.

1.7.3. Security and Privacy

The current version of the positioning model does not address security or privacy concerns related to the shared information; therefore, malicious users could potentially utilize the model to obtain other users' device information, detect or track the users, or even deceive them with false information.

However, the model follows the line of peer-to-peer (P2P) networks file sharing, which are also highly dependent on its users' intent, but still have been proven to be relatively safe for most of the users' operations (specifically file sharing), although malicious users are always a threat. Nonetheless, ensuring the security and privacy of the shared information is a challenge that should be addressed as part of future work.

1.8. Document Organization

The next section shows the concepts required to understand the scope of this work. Section 3 shows the related work. The proposed context-aware positioning model (CAMPOS) is detailed in Section 4. The simulation scenario and the experiments are specified in Section 5. The results are shown and discussed in Section 6. Section 7 presents the conclusions and future work.

Chapter 2: Background

This section introduces the concepts and background information necessary to understand this thesis work. First, the context-aware computing paradigm is introduced. Then, the concepts behind positioning are briefly explained and a classification of positioning techniques is presented. Finally, the concepts of mobile ad hoc network (Manet) and opportunistic network (Oppnet) are explained, since they are the most frequent communication infrastructure that is available to support this type of collaboration process.

A brief summary of this section is provided in [115], including a comparison of these methods regarding the type of activities to be supported.

2.1. Positioning

The *position* of a resource is a representation of its physical location in a given environment. *Positioning* is the process of determining the position of the resource in such an environment. Positioning can be roughly divided in two categories, *outdoor* and *indoor*, based on the type of environment they address.

In outdoor environments, the GPS, a satellite-based positioning system, is currently the most widely used. It offers maximum coverage for positioning with relatively little effort [70]. However, its performance is limited on indoor environments, mainly due to obstructed line-of-sight (LOS) transmission between receivers and satellites, which severely affect GPS' accuracy.

On the other hand, as we mentioned earlier, indoor environments tend to be more complex than outdoors [87] due to the presence of several types of obstacles, which carry about issues with signal readings, and thus to a higher error on the positioning estimations.

As we mentioned earlier, some authors believe that outdoor positioning has a concrete solution in the form GPS [59]. However, the appearance of new technologies and the proliferation of wireless and mobile networks has allowed for positioning to remain an open area, filled with research opportunities [59, 94, 131]. Important positioning research issues include the degree of accuracy of the positioning estimations; the delay of the estimations; the amount of concurrent requests that can be processed simultaneously; and the coverage area of the positioning service; the energy consumption of the positioning process; and the reliability of positioning services, i.e., the degree of availability of these services.

Moreover, different types of positioning information could be used, depending on the positioning requirements and limitations of the user's application. According to Hightower

et al. [69], the main types of positioning are *physical*, *symbolic*, *absolute*, and *relative*. Physical positioning is expressed in the form of coordinates, which identify a point on a multi-dimensional map (i.e., 2-D or 3-D). The most widely used coordinate systems are degree/minutes/seconds (DMS), degree decimal minutes, and universal transverse mercator (UTM). Symbolic positioning expresses a location in a natural-language way, such as “in the office”, “in the third-floor bedroom”, etc. Absolute positioning uses a shared reference grid for all located objects. Relative positioning depends on its own frame of reference, and its information is usually based on the proximity to known reference points or base stations [94].

The following subsections include a review of the most widely used strategies for positioning. In general, most positioning methods attempt to measure one or more signals, and then process these measurements in order to estimate the position of a resource. A positioning system uses different kinds of signals and varied technologies to determine a resource’s position, depending on the technologies used [69]. These technologies can be categorized in four groups: *Infrared*, *radio frequency*, *ultrasound*, and *inertial*, with radio frequency signals being the most popular [137]. We will not address these technologies any further in this thesis, because they are out of the scope of this research. Instead, we focus on the communication and positioning techniques that make use of these technologies (e.g., WiFi, RFID, etc.).

Based on the information measured and how the position estimation is performed, we can classify positioning techniques in four groups: (1) *angulation* and *lateration*, (2) *proximity*, (3) *fingerprinting*, and (4) *scene analysis*. Angulation, lateration, fingerprinting, and scene analysis can provide absolute, relative and proximity position information, while proximity only provides proximity information. Next we explain each of them.

2.1.1. Angulation and Lateration

Angulation and lateration use the geometric properties of triangles (i.e., angles and distance, respectively) and a set of reference points with known locations to estimate the position of a resource. The accuracy of this positioning approach improves when more reference points are used for the estimation process. An advantage of this method is that it involves a small setup effort in order to start calculating the resources location.

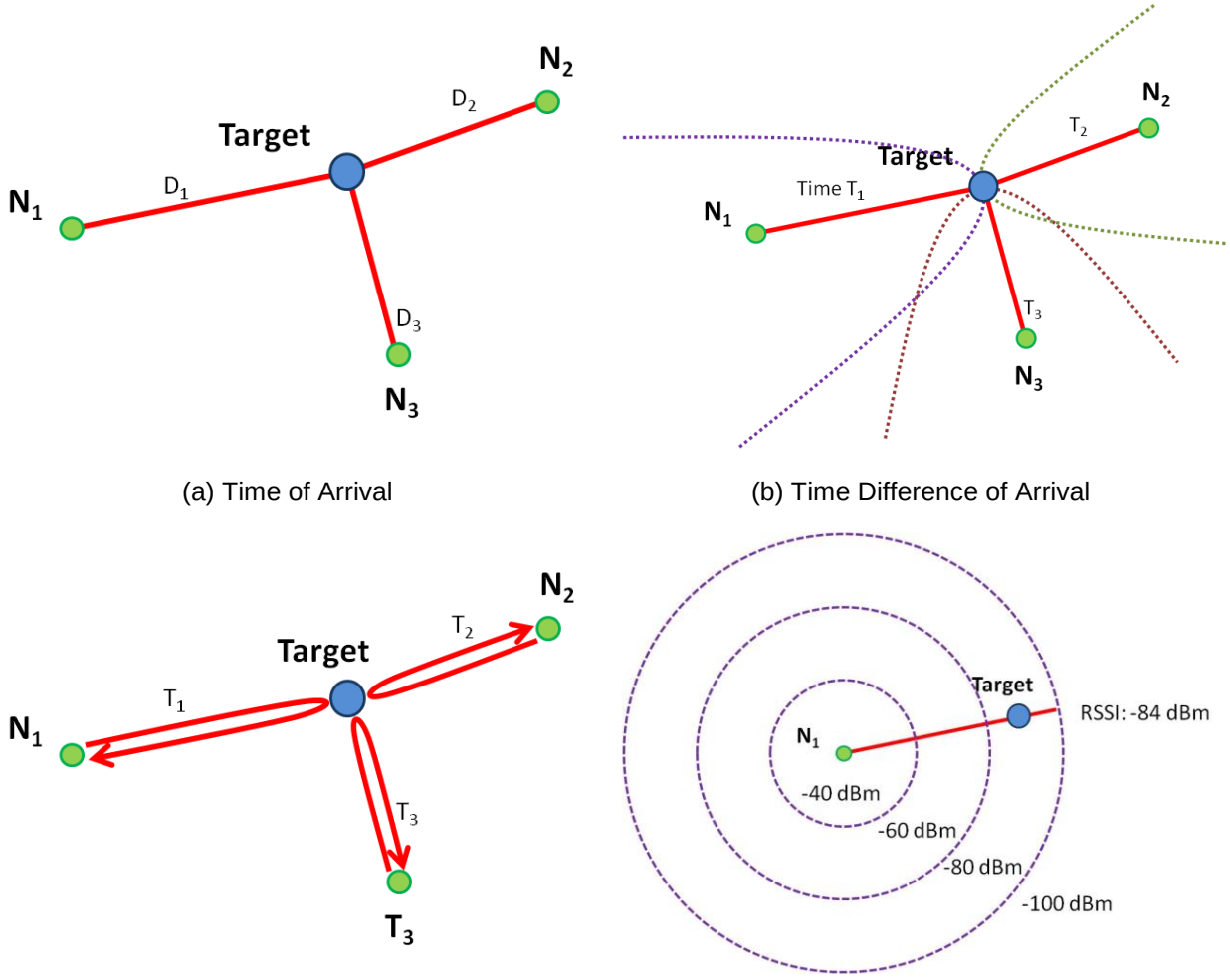
2.1.1.1. Lateration

Also known as range measurement, it estimates the position of a resource measuring its distance to at least three reference points with known geographical coordinates. Using the direction or length of the vectors drawn between the location to be estimated and the reference points, the absolute position of the desired resource can be calculated [162]. Five methods are commonly used to estimate positions using lateration: *time of arrival*, *time difference of arrival*, *round-trip time of flight*, *received signal phase*, and *received signal strength*. *GPS*, a special case of lateration method, and it is also addressed in this

section.

Time of Arrival (TOA)

TOA assumes that the distance between two devices is directly proportional to the time it takes to send and receive a message. TOA-based positioning systems measure this time, and use it to calculate the distance between transmitter and receiver. Figure 2-1-a shows three devices (N_i), each located at a distance $D_i$from the target. The time it takes for a response from these devices to reach the target determines the distance between them, and this information is used to estimate the target's position.



(a) Time of Arrival

(b) Time Difference of Arrival

(c) Round-trip Time of Flight

(d) Received Signal Strength Intensity

Figure 2-1. Lateration methods I

At least three reference points are required for 2-D positioning [49], and an additional one for 3-D. In order for TOA to work, the clock of all participants has to be precisely synchronized, and a time-stamp must be labeled in the message, so that the measuring unit can estimate the distance the signal has traveled.

Time Difference of Arrival (TDOA)

TDOA determines the relative position of a resource by examining the time difference at

which its signal arrives at multiple measuring units. Thus, a target position can be estimated from the intersections of two or more TDOA measurements. For each of these measurements, the target must lie on a hyperboloid with a constant range difference between the two measuring units. Two hyperbolas are formed from TDOA measurements at three fixed measuring units to provide an intersection point, which locates the target resource [49]. Note that the receivers do not need to know the absolute time at which the pulse was transmitted; only the time difference is relevant.

Figure 2-1-b shows the way in which TDOA performs the position estimation. The three dimensional hyperboloids (dotted lines), formed from each device N_i, represent the possible locations of the target given their time differences T_i measured from communications with the target. The intersection from these hyperboloids forms a three dimensional polygon that is assumed to include the target true location.

Round-tripTime of Flight (RTOF)

This method measures the time it takes a signal to travel from the target device to the measuring unit, and back [61]. RTOF requires a less strict relative clock synchronization than TOA, although both methods use the same range measurement mechanism.

The measuring unit can be considered as a common radar, with the target responding to an interrogating radar signal and the complete roundtrip propagation time being calculated by the measuring units. However, it is still difficult for a measuring unit to know the exact delay/processing time it takes the target to return the signal. This delay can be ignored in long or medium-range systems, if it is small in comparison to the transmission time. However, in short-range systems, such as those used for indoor location, it cannot be ignored. Figure 2-1-c shows the positioning strategy used by RTOF, which is essentially identical to TOA, with the exception that each of the T_i measurements accounts the two-way duration of a communication, instead of measuring only the one-way duration of a response (as TOA does).

Received Signal Strength (RSS)

Also known as Signal Attenuation, RSS estimates the position of a resource by measuring its distance from a set of measuring units based on the attenuation of emitted signal strengths [78]. RSS calculates the signal path-loss due to propagation, using theoretical and empirical models to translate the difference between emitted and the received signal strength into a range estimate. In Figure 2-1-d, the distance of the target from device N_1 is denoted by the RSSI (received signal strength intensity); depending on the power and frequency of the transmitter, this distance could range from 1 to several meters. When used jointly with RSSI readings from other devices, the target can estimate its position.

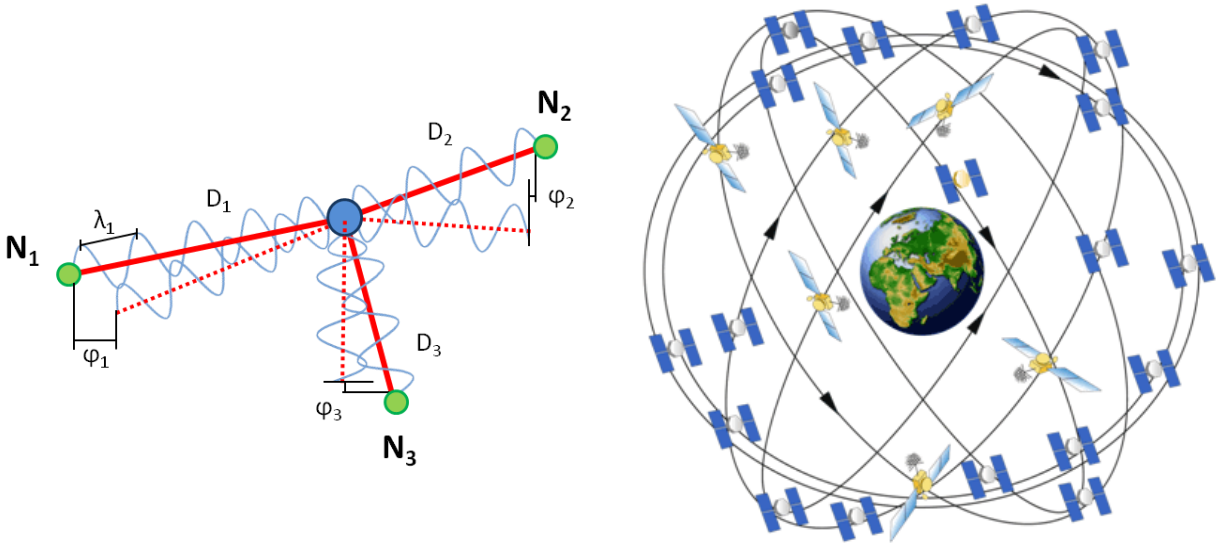
RSS requires an important setup effort, and it can be affected by multipath fading and shadowing effects, which are always present in indoor environments. This can be addressed by using additional reference points. Moreover, the size of the positioning grid greatly affects the accuracy. Smaller grid spacing provides greater accuracy up to a

certain point (values measured 15 centimeters apart will be more or less the same) [131]. On the other hand, increasing the size of the cells in the grid reduces the search space in the database, drastically decreasing the accuracy of the positioning.

Received Signal Phase (RSP)

Also known as Carrier Signal Phase of Arrival, this method uses the carrier phase (or phase difference) of a frequency range to estimate the position of a target [130].

In order to understand how RSP works, let us assume that all participating devices emit sinusoidal signals of the same frequency with a zero phase offset, as shown on Figure 2-2-a. The RSP method calculates the phase difference of all signals received at the target, estimating its position based on those calculations. For an indoor positioning system, it is possible to use the signal phase method together with TOA/TDOA or RSS method to increase accuracy. However, this method requires a direct line-of-sight signal path; otherwise, it will cause more errors, especially in indoor environments.



(a) Received Signal Phase (b) Global Positioning System

Figure 2-2. Lateration methods II

Global Positioning System (GPS)

GPS is a satellite-based positioning system (Figure 2-2-b), currently the most widely used in outdoor environments because it provides maximum coverage. GPS capability can be added to various devices simply by adding GPS cards and accessories to these devices. This enables position-based services, such as navigation, tourism, etc. [70]. However, GPS cannot be deployed for indoor use, because LOS transmission between receivers and satellites is not possible in an indoor environment.

A GPS receiver calculates its position by precisely timing the signals sent by GPS satellites high above the Earth. Each satellite continually transmits messages that include

the time the message was transmitted, and the satellite position at time of message transmission. The receiver uses the messages it receives to calculate the transit time of each message and to determine its distance to each satellite. These distances and the satellites positions are used to compute the position of the receiver [160].

At least three satellites are required to calculate a target's position, since space has three dimensions and it is assumed that the target is near the Earth's surface. However, even a tiny clock error, multiplied by the speed at which satellite signals propagate, results in a large positional error. Therefore, receivers usually employ four or more satellites to resolve their position, although fewer satellites can be used in special cases. If a positioning variable is already known (i.e., altitude), a receiver can compute its position accurately using only three satellites.

When fewer than four satellites are accessible, some GPS receivers may use additional clues or assumptions (such as reusing the last known altitude, dead-reckoning, or inertial navigation) to give a less accurate or degraded position estimation [22].

GPS also has some disadvantages; for instance, its accuracy depends on the number of visible satellites; its setup time can be quite long, many minutes in the worst case; and power consumption can be high. Moreover, GPS does not work indoor or when satellites are in shadow [158].

2.1.1.2. Angulation

Also known as direction of arrival, angulation calculates the position of a device by computing the angles relative to two or more reference points with known geographical coordinates. Using the angle of the vectors drawn between the target's location and the reference points, it calculates the absolute position of the desired resource, as shown in Figure 2-3 [162]. The most well-known method used for angulation is the *angle of arrival* (AOA).

In AOA, the location of a target resource can be estimated using the intersection of several pairs of angle direction lines, each formed by the circular radius from a base station or a beacon station to the mobile target. AOA methods use at least two known reference points and two measured angles to derive the 2-D location of the target resource. This estimation, commonly referred to as direction finding, can be accomplished either with directional antenna or with an array of antennae [30].

The advantages of AOA are that a position estimate may be determined with fewer measuring units than lateration, three or two for 2-D environments, and three for 3-D. Also, no time-synchronization between measuring units is required. Its disadvantages include relatively large and complex hardware requirements, as well as location estimate degradation as mobile targets or measuring units move farther from each other.

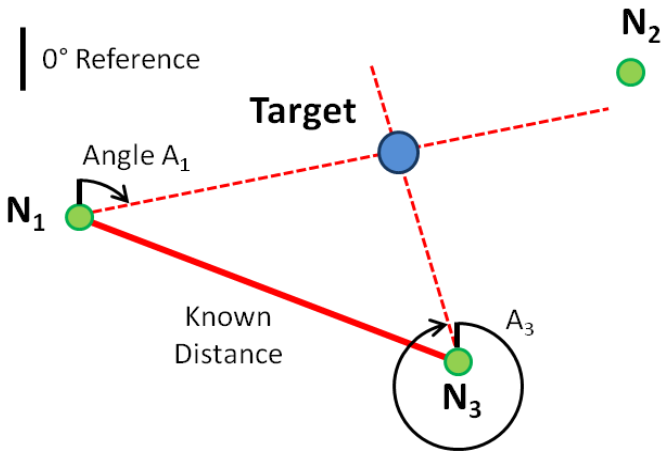


Figure 2-3. Angulation by Angle of Arrival

2.1.2. Proximity

Proximity positioning usually relies upon a dense grid of detectors, each with a well-known position. When a mobile device is detected by a single antenna, it is considered to be collocated with it. When more than one station detects the mobile target, it is considered to be collocated with the one that receives the strongest signal [17], or at the intersection area of both stations. The accuracy of proximity positioning systems depends on which detection technology is used, and on the number of reference points deployed in the physical environment. The greater the density of reference points, the higher the accuracy.

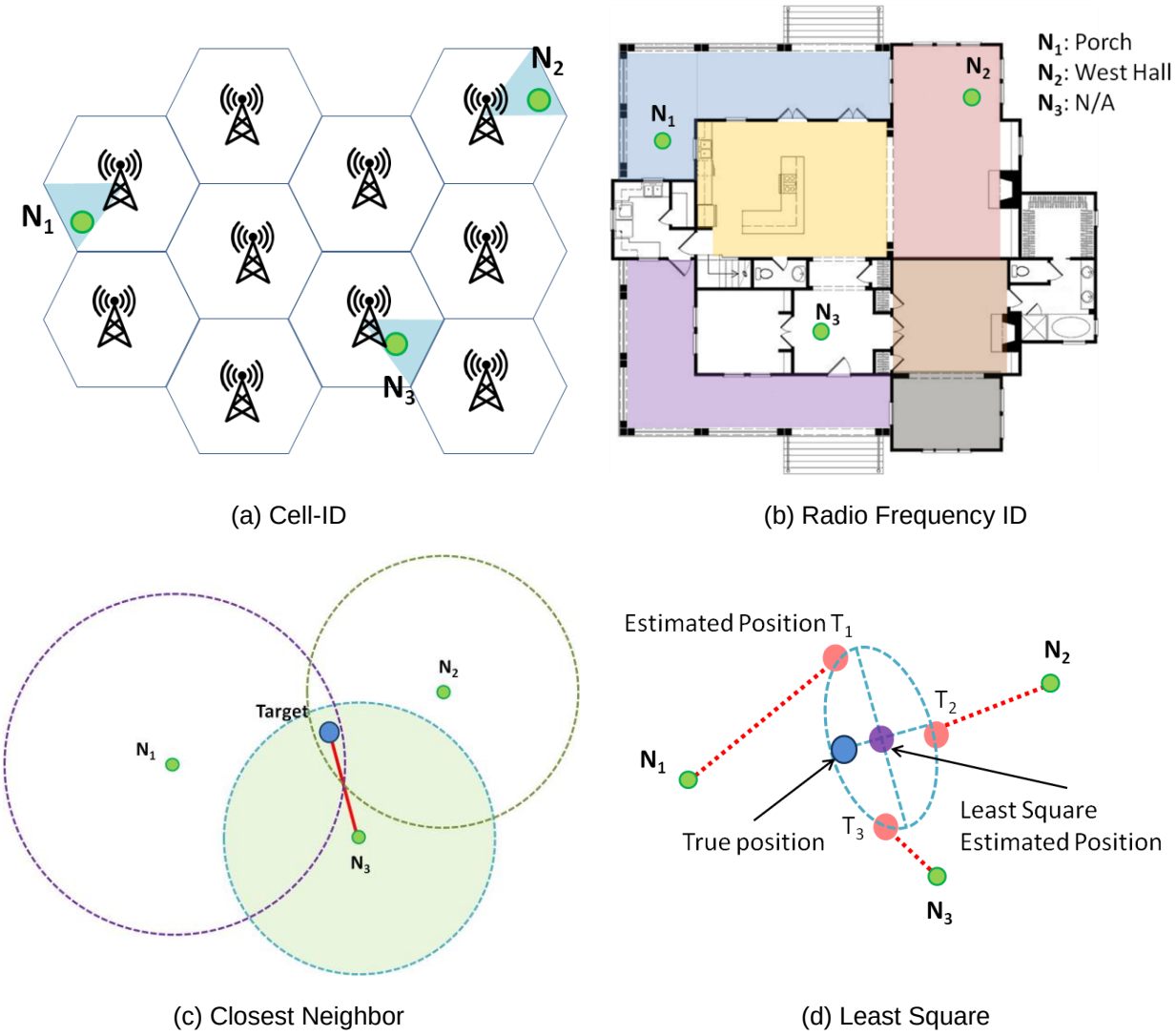
This method is relatively simple to implement over different types of physical media, although an important setup effort is required on early deployment stages. Positioning systems using infrared radiation (IR) and radio frequency identification (RFID) are often based on this method. Five methods have been considered for proximity positioning: *Cell ID*, *radio frequency identification*, the *closest neighbor algorithm*, and the *least square algorithm*. Next we explain all of them.

2.1.2.1. Cell-ID

Cell ID (CID), also known as Cell of Origin, relies on the fact that mobile cellular networks can identify the approximate position of a mobile device by knowing which cell site the device is using at a given time [158]. A base station covers a set of cells, each with a known position and identified by a unique Cell-ID. Cells are grouped into clusters, each of them identified by a Location Area Identifier (LAI). A mobile target continuously selects a cell and exchanges data with its corresponding base station. In turn, each station broadcasts both the LAI and the Cell-ID to its cells. Figure 2-4-a shows how the CID antennas provide to the shown devices, with high-grain positioning capability.

Since the mobile targets are always receiving these broadcast messages, they always know their corresponding Cell-ID. This allows the mobile targets to approximate

their position using the geographical coordinates of their corresponding base station, independent of the target's absolute position within the cell. The main benefit of Cell-ID is that it is already in use today, and it can be supported by all mobile handsets.



(a) Cell-ID

(b) Radio Frequency ID

(c) Closest Neighbor

(d) Least Square

Figure 2-4. Proximity strategies

2.1.2.2. Radio Frequency Identification

The radio frequency identification (RFID) is a means of storing and retrieving data through electromagnetic transmission to a radio-frequency compatible integrated circuit. RFID enables flexible and cheap identification of individual person or device [33], and it is commonly used in complex indoor environments, such as offices, hospitals, etc.

There are two kinds of RFID technologies, passive and active RFID [59]. With passive RFID, a tracked tag is only a receiver, making them small and inexpensive at the cost of a reduced coverage range. Active RFID tags are transceivers, actively transmitting their identification and other information, which makes the cost of tags higher, but it provides a greater coverage area of active tags.

In Figure 2-4-b, a floor-plan of a house covered with RFID sensors is presented, showing three devices N_i carrying RFID tags. Devices N_1 and N_2 are in contact with RFID sensors, which means that their positions are known (high-grained, defined by the location of the sensor), whereas device N_3 is at a blind spot, and thus its position is unknown.

2.1.2.3. Closest-Neighbor Algorithm

The closest neighbor algorithm (CN) can be used to approximate the position of a device in a fixed environment. The process, shown in Figure 2-4-c, is straightforward. In order to understand how CN works, let us consider a group of base stations arranged in a regular grid, in which a station is located L meters away from each other. In order to determine the position of a particular resource, each station performs a distance measurement to the resource in question.

Let d_i be the distance measurement to each neighbor performed by base station i, which is located at $R_i = [x_i, y_i]$. The CN algorithm estimates the position of the target resource as that of the station that is located closest to it. In other words, the position of a resource is the value of R_i for which the corresponding distance measurement, d_i is the minimum in the set [82].

2.1.2.4. Least Square Algorithm

This method focuses on minimizing the value of the objective function $f(x) = \sum_{i=1}^{N}\left(\sqrt{(x-x_i)^2 + (y-y_i)^2 - d_i}\right)^2$, where N is the number of reference base stations. The square-root term is the distance between a point (x, y) in the Cartesian coordinate system and a reference base station located at the point (x_i, y_i), and d_i is known as the residual of the estimate. Given that this is a minimizing function, the closer it approaches to the target's position, the lower the function's value would be. At $f(x) = 0$, it would have estimated the target's actual position.

However, in practice, the set of distance measurements, $d_i\ (1 \leq i \leq N)$ always contains errors, so the function will never be zero even at the target's position. These errors are related to synchronization mismatches between the transmitter and receiver devices, (known as systematic error), or due to obstructed line-of-sight channel conditions, known as channel-related errors [82], pictured in Figure 2-4-d. These channel conditions generally result in the strongest signal being received with longer delay, making the distance measurement longer than it should be, thus decreasing accuracy.

2.1.3. Fingerprinting

Also known as *scene analysis*, this technique calculates the position of resources in a bounded physical space by comparing the current measurements of a given set of signals with pre-measured data related to particular locations. Typically, it involves two phases: an offline training phase, and an online estimation phase.

During the offline phase, samples of location related data (e.g., Wi-Fi received signals strength) are collected for the whole physical space considered for the estimation process. During the online stage, the currently observed signal strengths of a resource are used in conjunction with the previously collected data to figure out an estimated position for the target resource.

The fingerprinting technique is simple to deploy compared to AOA or TDOA techniques [81], but it is costly to implement over a large area. Instead of depending on accurate estimates of angle or distance to determine the location, location fingerprinting associates location-dependent characteristics (such as signal attenuation) with a location and uses these characteristics to infer location position.

Additionally, fingerprinting is quite accurate, but it involves an important effort to collect samples during the offline phase [162]. One of the main challenges to this technique is that the signal emitted by the resources could be affected by diffraction, reflection, and scattering in indoor environments.

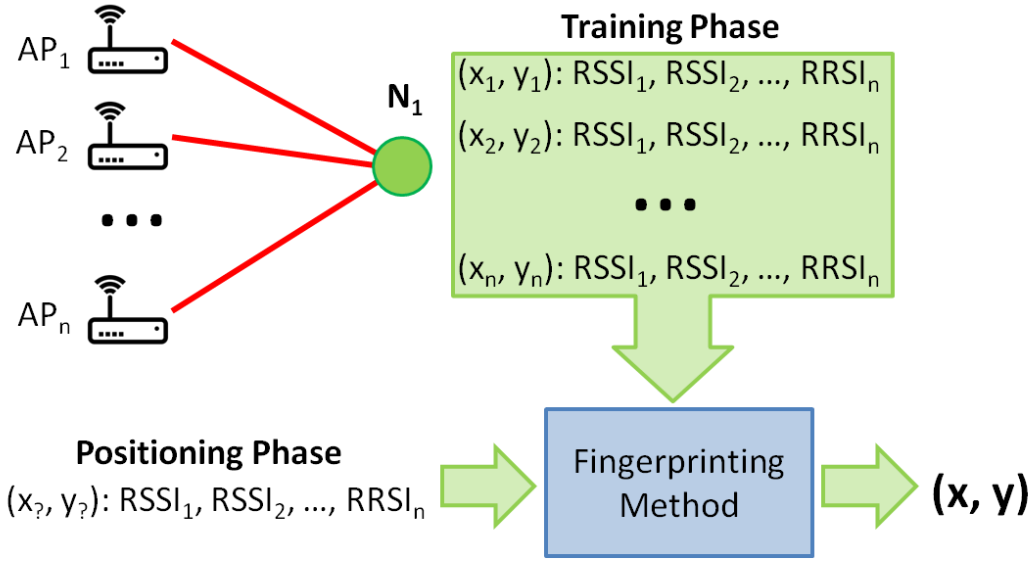


Figure 2-5. General scheme of fingerprinting

Fingerprinting can be performed by using pattern recognition based methods and probabilistic methods, and the general scheme of the strategy is shown in Figure 2-5. The main idea is to use sets of received signal strengths ($RSSI_i$) from several access points (AP_i), each with their respective intensities, and associate them to fixed cells in a grid (x_k, y_k). This data is used during the training phase to assemble a "directory" of positions based on perceived RSSIs, and then during the positioning phase, a device uses its perceived set of RSSI to determine its position.

The following subsections provide an in depth explanation of the methods used to perform both pattern recognition and probabilistic fingerprinting.

2.1.3.1. Pattern Recognition-based fingerprinting

Pattern recognition methods are applied to fingerprinting positioning, which means they are intended for indoor environments. These methods include the *k-nearest neighbor*, *support vector machines*, *smallest m-vertex polygon*, and *neural networks*.

k-Nearest Neighbor Averaging Method (kNN)

The kNN averaging uses the online signal strength to search for k closest matches of known locations in a signal space from a previously built signal database, by means of the root mean square errors principle [166]. In this approach, k is a parameter that can be adapted in order to improve performance. By averaging these k location candidates with or without adopting the distances in signal space as weights, an estimated location is obtained via weighted kNN or un-weighted kNN.

Support Vector Machine (SVM)

A widely used technique for data classification and regression, SVM is a tool for statistical analysis and machine learning. SVMs have been used extensively for a wide range of applications in science, medicine, and engineering with excellent empirical performance [37]. Both support vector classification of multiple classes and support vector regression have been successfully used in location fingerprinting, by treating the positioning problem as a classification problem [94]. Thus, the SVM attempts to classify the received signal strength of a target to multiple reference points into a class, i.e., a position in the grid.

Smallest M-vertex Polygon (SMP)

SMP uses the online signal strength values to search for candidate locations in signal space with respect to each signal transmitter separately. M-vertex polygons are formed by choosing at least one candidate from each transmitter (assuming a total of M transmitters). Averaging the coordinates of vertices of the smallest polygon (the one with the shortest perimeter) gives the position estimate of a target resource [94].

Neural Networks

Usually, a multi-layer perceptron (MLP) network with one hidden layer is used for neural-networks-based positioning system [166]. During the offline stage, the signal strength and the corresponding location coordinates are adopted as the inputs and the targets for the training purpose. After the training stage, appropriate weights are obtained. The input vector of signal strengths is multiplied by the trained input weight matrix, and then added with input layer bias, if a bias is chosen. The result is put into the transfer function of the hidden layer neuron, and the output of the function is multiplied by the trained hidden layer weight matrix, and then added to the hidden layer bias if it is chosen. The output of the system is a two-element vector for 2-D or a three-element vector for 3-D estimated location.

2.1.3.2. Probabilistic fingerprinting

These methods attempt to estimate the probability of a resource being at a certain place,

given the observed measurements at each location [85]. Connections and divisions between different places can be considered, since someone cannot walk through walls, or through water bodies. This approach is more complex and requires more computational power, but it usually presents better results than the previous ones. The most commonly used method for probabilistic fingerprinting are the *Bayes' Theorem* and *Markov Chains*.

Bayes' Theorem (BT)

This method addresses positioning as a classification problem [85]. Assuming that there are n location candidates $\{L_1, L_2, L_3, \dots, L_n\}$, and s is the signal strength vector observed during the online stage, the following decision rule can be obtained: Choose L_i if $P(L_i|s) > P(L_j|s)$, for $i, j = 1,2,3, \dots, n; j \neq i$. Here, $P(L_i|s)$ denotes the probability that the mobile node is in location L_i, given that the received signal vector is s [94]. Moreover, let us assume that $P(L_i)$ is the probability that the mobile node is in location L_i. The given decision rule is based on posteriori probability. Using Bayes' formula, and assuming that $P(L_i) > P(L_j)$ for $i, j = 1,2,3, \dots, n$, we choose L_i if $P(s|L_i) > P(s|L_j)$, for $i, j = 1,2,3, \dots, n; j \neq i$, based on the likelihood that $P(s|L_i)$ is the probability that the signal vector s is received given that the mobile node is located in location L_i.

Markov Chains Positioning (MC)

The key idea of Markov Chains positioning is to compute and update a probability distribution over all possible locations in the environment [23]. Let $l = \langle x, y, \Theta \rangle$ denote a location in the state space of a target resource. The distribution, denoted by $P(L_t = l)$ expresses the target's subjective belief for being at position l at time t. Initially, $P(L_{t0})$ reflects the initial state of knowledge. If the target knows its initial position, $P(L_{t0})$ is centered on the correct location; if a resource does not know its initial location, $P(L_{t0})$ is uniformly distributed to reflect the global uncertainty of the resource. $P(L)$ is updated whenever new sensor readings are received, allowing for positioning. This method is usually combined with vision analysis techniques for robot navigation [23].

2.1.4. Scene Analysis

This technique analyzes images received from one or more capturing points (e.g., cameras located in the surveillance area) to attempt identifying one or more target resources [21]. Real-time analysis of images or video is feasible when a small number of objects are present; otherwise, it is more efficient to combine this technique with fingerprinting or proximity techniques. Using vision analysis involves an important effort during the setup phase, because they rely heavily on monitoring equipment.

For vision-based positioning systems, a low price camera can cover a large area, and the targets require no additional devices for the position estimation. While vision analysis has unique advantages over other positioning systems, it also presents unique challenges. Privacy is an issue due to the nature of vision analysis, where multiple images of the targets are acquired. Since the position estimations are based on the saved vision

information in a database, it needs to be updated if there is any change in the environment, like moving a desk from one side of the room to the other [59].

Vision-based positioning systems can also be greatly influenced by interference sources, such as weather, light, etc. For example, the turning on and off a light in a room reduces the detection accuracy of a target's position. A person's appearance in an image varies significantly due to posture, facing direction, distance from the camera, and occlusions [21]. Moreover, trying to position multiple resources moving around at the same time is still a challenge, due to the high computational requirements of this technique. Although a variety of algorithms can overcome most of these difficulties, a solution must work fast enough to make the system responsive to the room's occupants. The Simultaneous Localization and Mapping technique tries to address these problems.

Simultaneous Localization and Mapping (SLAM) addresses the problem of a resource (usually a robot) navigating an unknown environment. While navigating the environment, the robot seeks to acquire a map of its environment, and at the same time it wishes to localize itself using its map [109]. The use of SLAM can be motivated by two different needs: (1) detailed environment models, or (2) an accurate sense of a mobile robot's location. SLAM serves both purposes, but we will focus only on the positioning part. SLAM can be achieved through extended Kalman filters, graph-based optimization techniques, and particle filtering, among other navigation techniques. The general idea of SLAM is pictured on Figure 2-6.

Extended Kalman Filters (EKF)

Historically, EKF [103] is the earliest and perhaps the most influential SLAM algorithm. First, a map with all known landmarks must be stored in a database accessible to the robot. If the identity of an observed landmark is unknown, EKF cannot be applied. Typically, a robot compares which of the landmarks stored in the database most likely corresponds to a landmark just observed, using this information to estimate its current position. The proximity estimation to a landmark considers measurement noise and actual uncertainty using Mahalanobis distance [39], which is a weighted quadratic distance, to gauge similarity between observed and stored data. To minimize the chances of false data associations, many implementations use visible features to distinguish individual landmarks and associate groups of landmarks observed simultaneously [157]. A key limitation of the EKF solution to the SLAM problem lies in the quadratic nature of the covariance matrix. A number of researchers have proposed extensions to the EKF SLAM algorithms that gain remarkable scalability through decomposing the map into sub-maps, for which co-variances are maintained separately. EKF SLAM has been applied successfully to a large range of navigation problems, involving airborne, underwater, indoor, and various other vehicles [157].

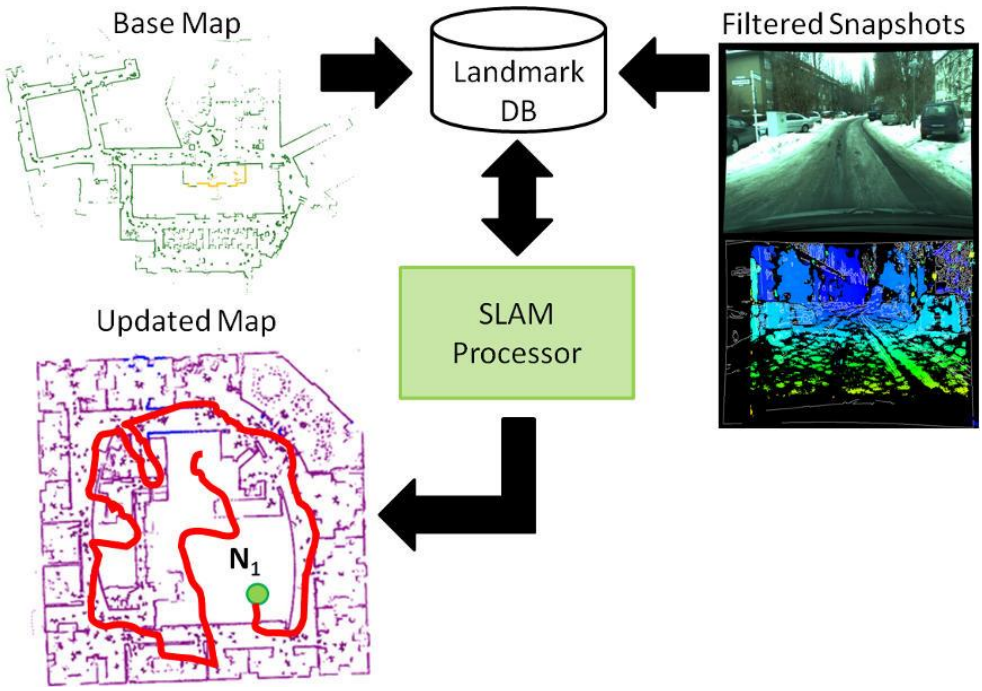


Figure 2-6. SLAM applied to robot navigation

Graph-Based Optimization (GO)

This method addresses the SLAM problem through nonlinear sparse optimization. Landmarks and robot locations can be thought of as nodes in a graph. Every consecutive pair of positions $\{x_t - 1, x_t\}$ is tied together by an arc that represents the information conveyed by the odometry reading u_t. Other arcs exist between locations x_t and landmarks m_i, assuming that at time t the robot sensed landmark i. Arcs in this graph are soft constraints. Relaxing these constraints yields the robot's best estimate for the map and the full path [98]. GO SLAM can scale to much higher dimensional maps than EKF SLAM. Unlike EKF, GOT does not use a covariance matrix, which translates into less used space and lower update times, depending on the size of the map [157]. Although the update time of the graph is constant and the amount of memory required is linear, optimizations can be expensive. Finding the optimal data association is suspected to be an NP-hard problem, although in practice the number of plausible assignments is usually small.

Particle Methods (PM)

This SLAM method is based on particle filters. In this paradigm, a particle represents a concrete guess of the value of the current state (position) of a robot based on observed landmarks. By collecting a set of particles, the particle filters capture a representative sample of the path distribution of the robot [109], allowing for an estimation of its position. Under controlled conditions, the particle filter has been shown to approach the true path as the particle set size goes to infinity. The key problem with this method is that the space of maps and robot paths is immense, exponentially scaling with the dimension of the underlying state space [110].

2.2. Context-Awareness

Context-awareness is a property that is potentially available in mobile devices and that is defined complementarily to location awareness. Whereas location may determine how certain processes operate around a participating device, context may be applied more flexibly with mobile users, especially with users of smartphones. Context-awareness originated in ubiquitous computing seeking to deal with the problem of linking changes in the environment with computer systems, which are otherwise static.

In the following subsections we will show other author's definitions of context and awareness, and then we will proceed to define what context-awareness is for the purpose of this thesis work.

2.2.1. Definition of Context

Context is a complex description of shared knowledge about physical, social, historical, or other circumstances within which an action or an event occurs [20], and it is necessary to understand the motive behind those actions and events. For example, if no context is given, the action of "shutting down" could refer closing a business, ceasing operations on a nuclear reactor, or even turning off a laptop.

According to Brézillon [19], context is "whatever does not intervene explicitly in a problem solving but constrains it", and it is continually evolving as the problem is solved. Context can be divided into external knowledge, which is not relevant for current tasks; and contextual knowledge, which is strongly connected to the task, although it is not considered as part of it. Schillit et al. [145] define context as the location and identities of relevant objects and people, and the changes that occur to them. Figure 2-7 shows an example of the importance of context, and how it affects not only the perception of objects and events, but also the behavior of the person interpreting the scene.

Ryan et al. [139] consider two additional elements, environment and time. Dey [41] defines context as the users' emotional state, focus of attention, location, orientation, date, time, and other users in their environment. Abowd [1] state that context is any information that can be used to characterize the situation of a person, place, or object that is considered relevant to the user-application interaction.

Messeguer et.al [107] address the problem of automating group awareness in computer-supported collaborative learning applications, by estimating group arrangements from location sensors and the history of the interactions of the participants. They propose a three-phase filtering strategy, which manages uncertain contextual information by identifying sources of uncertainty, representing uncertain information, and determining how to proceed based on such information.

Figure 2-7. Changes in the environment can alter the perception of identical objects

For purposes of this thesis work, we define context as *the elements that can help a device in the process of estimating its position at a given time*. This definition includes both elements in the environment and some that are part of the device itself. Specifically, it encompasses the positioning strategies, access points, and neighbors present in the device's vicinity; peripherals installed on the device (such as positioning transceivers and communication antennas); and status indicators (energy level and known positions).

2.2.2. Definition of Awareness

Awareness is a term used to denote knowledge created through the interaction of a resource and its environment, i.e., *knowing what is going on* [63]. It is meant to convey how individuals monitor and perceive the information in the environment they are in, which can be critical to the performance of the resource, and to the success of collaboration with other resources.

Dourish and Bellotti [44] define awareness as an understanding of the activities of others, which provides a context for your own activity, and it is used to ensure that individual contributions are relevant to the group's activity as a whole and to evaluate individual actions with respect to group goals and progress.

We define awareness as *the capacity of a device to detect and interact with elements in its environmental context*, namely other devices, access points, and positioning strategies.

2.2.3. Definition of Context-Awareness

Hull et al. [74] define context-awareness as the capacity of computing devices to detect, sense, interpret, and react to changes in the local environment and on the computing devices themselves. Salber et al. [142] define context-awareness as the ability to provide maximum flexibility to a computational service, based on real-time changes on its context. Ryan [140] defines context-aware applications as those that monitor input from environmental sensors and allow users to select from a range of physical and logical contexts based on their interests or activities. Fickas et al. [52] define context-aware applications (to which they refer to as “environment-directed”) as applications that monitor changes in context, and adapt their operation according to predefined or user-defined guidelines.

Following our previous definitions of context and awareness, we define context-awareness as the *capacity of a device to detect and interact with all the elements present in its vicinity that can help it in the process of estimating its position at a given time*.

2.2.4. Context-Aware Systems

A system can be considered as context-aware if it uses its context to provide relevant information and/or services to users, where relevancy depends on the user’s task [1]. In order for an application to be considered as context-aware, it must be able to “adapt to its context,” i.e., sense, interpret, and react to changes in context [134].

However, context-awareness is subjective and heavily dependent on the requirements of the consumer application. For example, a *GPS navigator* measures a car’s position and then adjusts it to fit in a map API, such as OpenStreetMap, showing the results to the users in a screen. The navigator is constantly updating the position of the car, and it “reacts” to events in the roads, such as corners and blocked roads, by suggesting changes in direction or speed.

Another example of a context-aware application is a *heart rate sensor monitor*. The sensor measures the rate of a patient’s heartbeats, and it sends such information to the monitor, which analyzes the historical and acquired data to determine whether the patient is in danger of a heart attack or not. If the monitor determines that the patient’s life is in danger, it sends an alert to the medical personnel at the hospital, or to an ambulance paramedics team, depending on the location of the patient.

Clearly, both applications are context-aware, but each utilizes different types of information as input, and reacts to different stimuli. Thus, in order to determine the scope and limitations of our proposal, we must delineate our working context, as well as the degree of awareness that our model provides to the users.

2.3. Mobile Ad-hoc Networks

Conventional cellular and mobile networks are, in a sense, limited by their need to count on preexisting infrastructure (i.e., base stations, routers) [57]. Mobile ad-hoc networks (MANETs), get rid of this limitation by allowing devices with similar characteristics to interact with each other without the need for centralized control. This does not mean, however, that they are without issue; traditional wireless communication problems such as bandwidth optimization, energy administration, and transmission quality, remain unsolved in MANETs [92]. In addition, due to the multi-hop communication and the lack of fixed infrastructure, new research problems have arisen, such as network discovery and maintenance, and ad-hoc addressing and self-routing.

Since mobile ad-hoc network topologies are wireless in nature, they are highly dynamic and random, given that nodes are free to enter and leave the network as they move around (Figure 2-8). Nonetheless, wireless communication has significantly lower transmission capacity than its wired counterpart, and since the data travels through the air, security is also an issue. Additionally, environmental factors can affect the transmission, causing higher data loss rates, additional delays due to retransmissions and jitter, channel overload from too many signals, etc.

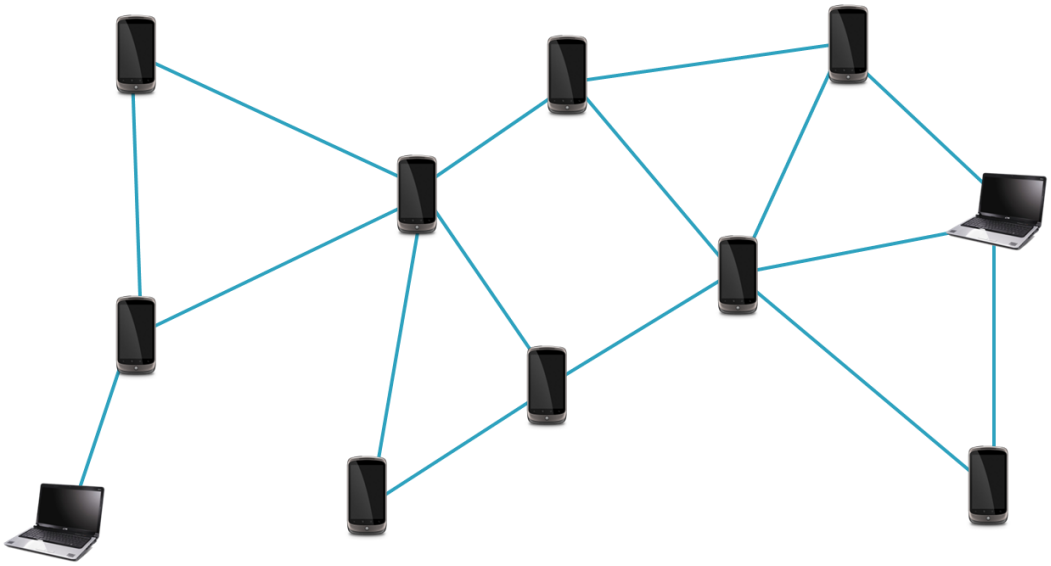


Figure 2-8. Example of a MANET topology

One of the more important problems of mobile ad-hoc networks is the energy consumption, which also affects the positioning process of a device. The highly mobile, wireless nature of the network nodes implies that they must rely on portable energy sources, such as batteries or other exhaustible means of power, and thus have limited operating time. Thus, energy saving must be an important criteria when designing mobile applications [57], and nodes must be power-aware, offering sets of functions according to their available power, in order to extend their operating time.

MANET configurations have been successfully applied in several fields, including commercial advertising, collaborative messaging, military tactical communication, sensor networks, rescue missions in times of natural disasters, as well as other commercial and educational applications.

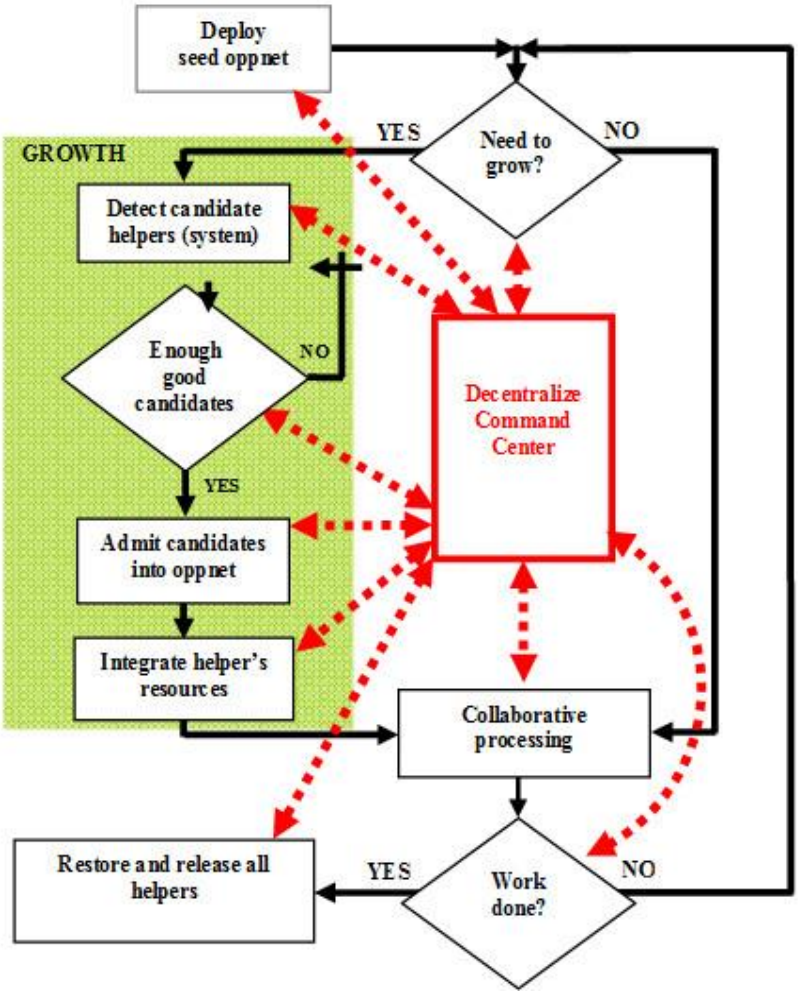


Figure 2-9. Example of a flowchart of an OPPNET [93]

2.4. Opportunistic Networks

Traditional networks are deployed together with fixed, pre-defined network parameters, such as size, location, etc. OPPNETS, on the other hand, are a superset of MANETs that contain sensor networks [93], which can be deployed on the fly. After deployment, the sensor nodes begin detecting the presence of nodes or systems from different communication media, such as Bluetooth, wired internet, WiFi, Radio, RFID, Satellite, etc. Any detected node or system, called “helpers”, is identified, classified, and evaluated, and any potential candidate that is useful and reliable is invited to join the network following the procedure detailed in Figure 2-9.

Candidate helpers can either accept or reject the invitation. Only those helpers that accept are admitted into the network. At the time of joining, their resources are integrated into the OPPNET, and any task being processed can be offloaded to, or distributed amongst, the new members.

A human operator, and/or an autonomous decentralized command center determines the tasks performed by an OPPNET throughout its lifetime. When its goals are reached, the OPPNET releases its helpers and restores them to a state that is closest to when they were invited to join the network, minimizing intrusiveness and potential damage to its systems. Thus, OPPNETS can be used as a bridge between disjoint communication media, leverage available services, and also provide access to sensor y data from diverse sensing systems that are already present in our environment, without the need to deploy additional instrumentation.

Chapter 3: Related Work

Several proposals address the problem of positioning on mobile, heterogeneous outdoor environments. Most of these approaches commonly rely on communication between a target resource (i.e., a mobile device) and a series of reference points with known positions (e.g., GPS satellites, access points, or other mobile devices) to estimate the position of the intended target through autonomous or collaborative approaches. These research works are usually focused on finding alternatives to GPS, or on improving the accuracy of positioning estimations by experimenting with hybrid techniques or using novel approaches.

Although some of these works are able to reduce the error of a positioning estimation to the range of centimeters, they typically require an existing communication infrastructure and/or a source of energy to function properly, both of which are usually not necessarily available in mobile environments. In addition, most of the pre-established proposals address the positioning problem focusing on a single positioning strategy (or a static combination of various techniques), and they tend not to take advantage of contextual elements present in the devices' environment.

Next we present the different alternatives by grouping the related work into four categories; self-positioning, collaborative positioning, context-aware positioning, and analog studies.

3.1. Self-Positioning

Self-positioning is the positioning process performed by a device through its own means, or by accessing a positioning strategy embedded in the scenario, without help from its peers. Several proposals have been reported to provide devices with access to this service. For instance, Shin et al. [152] propose a personal indoor/outdoor Wi-Fi Positioning System using received signal strength (RSS) from dense Wi-Fi access points (AP) dedicated to localization of Android-based smartphones. This method measures the RSS from each AP three times, calculating the mean value and comparing it to each training value (stored in a database). If the difference between the mean value and the training values is below a given threshold, the training value is withdrawn and the mean of filtered training values is calculated again. Finally, the mean value is compared with the database value and a proper location on the map is found with a proper scan time and threshold, thereby yielding a low error rate.

Other possibility to perform self-positioning is to use the Global Navigation Satellite Systems (GNNS), which incorporates several satellite-based positioning systems, including GPS, GLONASS (GLObal NAvigation Satellite System), Galileo, and Compass [71]. In GNSS, the measurements obtained from a group include extra measurements, specifically the relative distances between the neighboring users. As a result, there are

several factors that impact on the positioning accuracy, and cause problems for the performance analysis, for instance clock bias or synchronization issues, and geometric position dilution of precision (i.e., the multiplicative effect of satellite geometry on perceived precision, which affects the final error measurement over the course of several estimations). However, GNNS-based positioning systems remain the most widely available, due to their huge coverage area (which spans from entire countries to the whole world).

Zhang et al. [171] present an alternative approach in which a database (DB) is trained using navigational data from users, utilizing different strategies to survey sensor-based navigation errors in indoor positioning. They combine different navigation trajectories moving in and out of a target building to create the DB, restricting the time length of available indoor navigation trajectories, i.e., the trajectories from the last received GNSS signal before entering an indoor area, until the next GNSS signal is received after walking out of the target building. Their results showed that the errors in pedestrian navigation were reduced from 6.0 to 5.7 m during 5–10 minutes of indoor walking. Their study is focused on providing indoor positioning, and outdoor signals are used only to establish entry and exit points throughout the user's indoor route.

Evenou and Marx [48] present a dead-reckoning navigation structure and signal processing algorithms for self-localization of an autonomous mobile devices. Their proposal fuses pedestrian dead-reckoning and WiFi signal strength measurements to estimate the position and bearing of mobile devices. Although dead-reckoning estimations derive in high accuracy errors, the WiFi measurements can help correct the drift while navigating urban areas with access points present, providing high-accuracy real-time navigation. They also propose a structure based on a Kalman filter and a particle filter, which allows to further improve the Wi-Fi measurements, and the accuracy estimations, by reducing the error to less than 2 meters of the target true position.

Shaw et al. [151] apply the problem of mapping an estimate of a user's current location to a semantically meaningful POI, such as a home, restaurant, or store. To this end, they propose a spatial search algorithm that infers a user's location by combining aggregate signals mined from billions of foursquare check-ins, with real-time contextual information. This work uses machine learning techniques to create an optimal ranking function, which learns from large amounts of implicit feedback, evaluating the performance of their system in a variety of real-world situations. This novel work takes advantage of social networks' positioning information to locate target devices. However, given the nature of the algorithms used, it requires an extensive location database.

The scheme proposed by Ning et al. [118] focuses in high accuracy localization of mobile stations in outdoor environments using a two-phase positioning method. During the training phase, they partition the target area into clusters from historical RSS data using an improved clustering scheme. The location of the assets can be estimated during the online phase, by analyzing these clusters with a refined intersection approach. Since

clusters are created with RSS deviations from the observed path loss model rather than raw RSS, the clusters remain invariant to changes in the base station's transmitter power. Thus, the intersection approach used in the online phase improves the accuracy of the location estimations.

Xu et al. [168] remarks that satellite signals from GNSS are vulnerable and susceptible to blockage in certain urban scenarios, and therefore they propose a system architecture that integrates WLAN fingerprinting, visual positioning, baroceptor altitude estimation, and GNSS to allow seamless indoor/outdoor positioning of vehicles and pedestrians. Their work is based on GNSS for outdoor positioning, and fingerprinting and SLAM for indoor positioning, with the possibility of alternating between these types of scenarios. Results indicate that they can indeed achieve seamless indoor/outdoor positioning with better accuracy than single-source methods, reaching accuracy levels of less than 1 meter for outdoors, and less than 3 meters for indoors.

3.2. Collaborative Positioning

Collaborative positioning proposals estimate the position of a target device by borrowing positioning information from neighboring devices, or by using multiple estimations from different sources (or from the same source).

Huang et al. [73] studied the multiplicative effect of various factors on the accuracy of GNSS collaborative positioning algorithms, using their proposed Collaborative Dilution of Precision model. Using such a model, they also analyze the performance of GNSS standalone positioning. Their results show that GNSS collaborative positioning accuracy is always higher than or equal to that of GNSS standalone positioning.

Sahoo et al. [141] state that collaboration is highly essential for nodes to perform positioning efficiently in outdoor and indoor environments. They use beacon and anchor nodes with known positions to estimate the position of a target node in a distributive manner, employing localization algorithms that consider fading and shadowing effects. They show analytical methods that use probability distribution functions to find the probability of wrongly identifying a transmission arriving from a node with location information, reducing the localization error and thus providing more accurate positioning information to the target node. These researchers state that their algorithm works well even when only one beacon node is present. The algorithm performs calculations with low time complexity, which is suitable when memory and energy constraints are present. This work is similar to our own, with the difference that Sahoo et al. use probability distribution functions to determine viable neighbor signals, and their experiments utilize smaller scenarios (200 × 300 m^2) and a greater number of nodes (250–500) in order to achieve acceptable accuracy levels.

Savarese et al. [143] developed a two-phase distributed algorithm for determining the

positions of nodes in an *ad hoc* wireless sensor network. The algorithm assumes that some of the users know their positions (i.e., they act as anchors). The anchors broadcast their positions to their neighbors, allowing the latter to estimate their positions with a certain degree of confidence, which improves iteratively with each new transmission from the anchors.

This work provides collaborative positioning in outdoor environments to all participants of an *ad hoc* network. However, unlike CAMPOS, the beacon nodes' position is transmitted to all nodes in the network and then improved iteratively, which could significantly increase the energy consumption of the process and overload the communication link.

Capkun *et al*. [26] propose a distributed, infrastructure-free positioning algorithm that uses distance between mobile nodes to build a relative coordinate system. In this algorithm the nodes' positions are computed in two dimensions, thereby providing location information accurate enough to support basic positioning capabilities within the environment. The nodes know the position of their neighbors using the relative coordinate system, but they have no way to translate such a position to a geographic coordinate system, unless nodes with GPS capabilities are included in the network.

The proposal of Capkun et al. is interesting because it enables nodes in ad hoc networks to collaboratively determine the positions of each neighbor in relation to their own. However, it does not take into account the physical scenario of the devices, and thus it is unable to provide the actual geographical position of the nodes.

Kealy et al. [83] discuss the applicability of collaborative positioning algorithms and techniques for mobile terrestrial devices, presenting a range of qualitative and quantitative measurement information that supports collaborative approaches. They assess the cost-benefit of these measurements, determining the point of diminishing marginal utility for positioning, i.e., the point at which integrating additional information provides a negligible return to the positioning performance.

Jing et al. [80] propose a collaborative positioning solution for indoor pedestrian navigation that integrates measurements from various users via peer-to-peer ranging. They propose a particle-filter-based adaptive ranging constraint collaborative positioning (ARCP) algorithm, which integrates inertial measurements, map information, and relative ranging. Their approach improves the positioning accuracy and robustness of indoor environments by applying a selecting-and-weighting scheme to the ranging constraint on each user, based on their ranging measurements and network geometry. They indicate that ARCP improves accuracy measurements by over 60%, while reducing positioning error in 45%, and providing enhanced robustness.

Jing et al. utilize Matlab mathematical simulations of real positioning an inertial navigation data applied to different types of indoor scenarios, analyzing the network

characteristics and the positioning outcome and comparing it to results obtained in posterior trials. Their simulated environments consisted of a square area of 100×100 m^2, with anchors randomly placed all around the simulation scenario.

3.3. Context-aware Positioning

Context-aware proposals prioritize reactive changes in their behavior to perform self-adaptation to changes in their surroundings. Although most collaborative approaches can be considered to be context-aware, in the sense that they “detect” their neighbors, true context-awareness requires adaption.

Ficco et al. [51] state that outdoor positioning has achieved a satisfactory degree of technological maturity and effectiveness, which is not the case for seamless indoor and outdoor exchange. They propose a hybrid location approach that switches among positioning technologies supported by the target device and that are available in the environment, in a dynamic and transparent manner. By combining RSS fingerprinting for indoor positioning and GPS for outdoor localization, their proposal performs opportunistic technology switching according to a count-and-threshold mechanism, providing ubiquitous location services across indoor and outdoor scenarios, as well as minimizing the power consumption of the device.

The work of Kealy et al. [83], first introduced in Subsection 3.2, assesses the cost-benefit of a wide range of qualitative and quantitative measurements of collaborative approaches, determining the point at which integrating additional information provides a negligible return to the positioning performance. This approach deviates from the traditional idea of greedily integrating all available signals of opportunity, stressing the need to identify an optimal set of measurements for the requirements of the application given its current context and need, i.e., “fitness for use”.

Wang et al. [164] present a discussion of RSSI-based positioning algorithms for indoor scenarios, proposing weighing schemes to leverage the credibility of the measured RSSI values, to cope with heavy signal strength distortion induced by to multi-path and shadowing. They also propose a boundary selection and local grid scanning to lower the searching time during online tracking, and RSSI data dissemination and collection schemes to reduce traffic overhead. Their results yielded accuracy values of up to 2 meters in 80% of the estimations; however, a few test cases showed an improbable error of up to one room.

3.4. Analog Studies

Eltahir [46] presents a study of the effect of some radio propagation models, such as free space and two-ray ground, on the communication capability of devices in simulated urban mobile ad hoc networks. The study shows that commonly used propagation models

do not necessarily provide results similar to real-world conditions. Thus, choosing an incorrect propagation model could translate into misleading positioning measurements, particularly when using Wi-Fi signal strength.

Similarly, Camp et al. [25] provide a detailed explanation of mobility models used on ad hoc network research, as well as an evaluation of their performance, and Stoffers et al. [154] provide a comparison of radio propagation models.

Oliver et al. [122] present a study aimed at exploring users' physical activity intensity, GPS speeds, and routes traveled during their activities by combining GPS, Geographic Information Systems (GIS), and accelerometry. Their work is relevant for our research due to their analysis of loss of GPS signals due to a range of methodological issues, such as low battery life, signal drop out, and participant noncompliance.

Baig et al. [6] present an overview of crowdsourced computer networks, which are network infrastructures built by citizens and organizations that are not necessarily involved with telecommunication enterprises. Community networks are a subset of crowdsourced networks, characterized for being open, free, and neutral. They are open because the knowledge to build them is available without restraint; free because the network access is non-discriminatory (i.e., anyone can join); and neutral because any technical solution available may be used to extend the network, and because the network can be used to transmit data of any kind by any participant, including commercial purposes.

Although all of the presented proposals deal with the positioning problem by either attempting to improve the accuracy of the estimations, or by providing novel approaches, only a few of them deal with the problem of providing positioning to devices without proper positioning capabilities, such as the proposals of Sahoo et al. [141] and Savarese et al. [143]. In addition, given that the main focus is to improve accuracy measurements, the energy consumption is generally overlooked when designing new positioning methods, unless the solution deals explicitly with either decreasing the consumption, or at least maintaining it within a certain threshold (e.g., the work of Sahoo et al. [141].

Chapter 4: Context-Aware Positioning Model

In order to address the problem of providing devices that are in any way positioning-impaired with means to determine their position, we have designed CAMPOS, a positioning model based on randomized decision trees (i.e., random forest). The model is capable of assessing a device's environment at any given time, essentially sensing its context (as defined in Subsection 2.2) in order to detect the presence of elements that can be used to provide positioning capability to the device. These elements include other devices, access points, and positioning strategies.

Once the context of a device has been assessed, the model processes the gathered information and determines which positioning strategies are available and accessible, ranking them based on their suitability to the device's sensed context. This ranking is performed by treating the decision problem as a classification problem, using a Random Forest classifier. The contextual variables are treated as classification features, and inputted to the random forest classifier, which provides the ranking as an output.

The model was designed to (a) allow users (i.e., stationary and mobile devices) to estimate their positions in outdoor environments, either by accessing positioning strategies directly, or by sharing positioning information with neighboring users having known positions; (b) allow a smooth transition when switching from using one positioning strategies to another; and (c) spend as little energy as possible, considering the positioning restrictions.

The model has been designed with a particular scenario in mind, specifically that of urban areas affected by disasters; however, it can be used in several similar scenarios, including university campus localization, vehicular theft prevention, etc. Additionally, given that the model is able to choose from a pool of several positioning strategies, the process of switching from one particular strategy to another during periods of intensive positioning queries should be transparent to the user; i.e., the model should manage to perform the switch in such a way that the users are unaffected, or at least not inconvenienced.

Finally, given that the model is based on sensing a device's context at intervals and communication over mobile ad-hoc networks, the energy consumption must be kept in check to assure that it does not negatively impact the device's autonomy during the whole positioning process.

Devices making use of the model would utilize their peripherals to sense their immediate environment, gathering contextual information that can be consumed by the model, in order to determine the suitability of each of the positioning strategies available in a given scenario. Thus, a device possessing any kind of positioning capabilities could either directly use a positioning strategy available in the scenario (i.e., perform self-positioning), such as GPS or radio frequency identification (RFID); or use the position of at least three of its neighbors as reference points, in order to estimate its position by itself

(i.e., collaborative positioning).

In a similar manner, devices with no positioning capabilities whatsoever, or those that possess positioning capabilities, but are in any way impeded from using them, could also take advantage of collaboration in order to estimate their position by using information from neighbors with known positions as input.

Thus, the model can provide devices access to at least one positioning strategy present in a given environment, enabling users to perform positioning in situations where they would normally be unable to, as well as favoring positioning strategies that would expend less energy than using the default strategy (usually GPS).

It is important to note that the model *does not* aim to improve the accuracy of the devices' positioning estimations. It only addresses the problem of providing context-aware positioning with reasonable energy consumption to devices participating in disaster relief efforts, or similar scenarios, regardless of whether these devices have the peripherals required to support such activity.

Additionally, although the model does not particularly address indoor environments, these scenarios are also supported, although with lower accuracy than in the intended outdoor scenarios. This accuracy decrement is variable from one indoor scenario to another (even between two consecutive measurements in the same scenario), and it is heavily dependent on the structure and materials of the building. Given that the model is optimized for outdoor use, it cannot manage the multipath and signal loss effects found inside buildings, rendering the collaborative part of the model useless due to high error (i.e., low accuracy).

However, the model can be used to determine which building a device is currently in when used indoors. This is useful to support several disaster relief activities, as well as other types of applications such as geo-social networking [7]. In disaster-relief effort scenarios, indoor positioning is typically used only to provide coarse-grained awareness information about the devices' location (e.g., first responders and equipment), given that the reference points required for these positioning strategies to work are usually unavailable or in a new location (e.g., in the case of collapsed buildings).

Therefore, if an outdoor positioning mechanism is able to provide coarse-grained positioning information when under a roof or under cover (i.e., with a relatively high error but within the building), the use of a more accurate, specific indoor positioning strategy, is usually discarded. Provided that the proposed model has such a capability, improving its support for indoor positioning is only desirable.

The following subsections provide an insight on the inner workings of the CAMPOS model. Section 4-1 shows an overview of the entire process. Section 4-2 deals with the first stage, the context sensing. The second stage, context information management, is presented in Section 4-3. The third stage, context-aware positioning, is shown in Section

4-4. Finally, we provide an example of the utilization of the model in Section 4-5.

4.1. Overview of the Proposed Model

Figure 4-1 shows an overview of each of the steps that encompass the model's stages: (1) sensing the context, (2) managing the context information, and (3) performing the context-aware positioning.

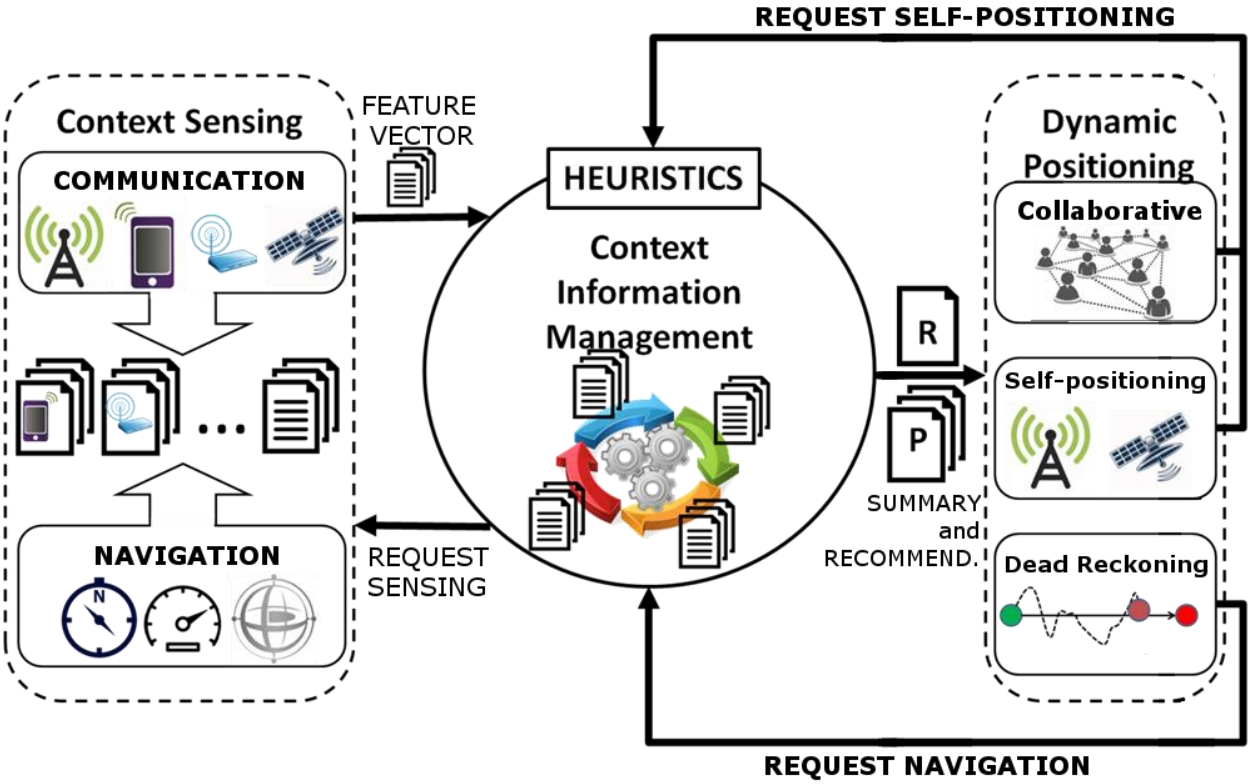


Figure 4-1. Three stage context-aware positioning model

During stage 1, the device "senses" its context through its peripherals, gathering information relevant to the positioning process from the scenario. Basically, the device sends a broadcast to other devices through all available channels (i.e., communication technologies), assembling a list of candidate reference points from neighboring devices and access points. This lists, coupled with the device's own positioning capabilities, is assembled as a feature vector and sent towards the second stage.

Over the initial phase of stage 2, the feature vector is received and processed, and irrelevant or duplicate information is pruned into a "summary" of the device's context. Then, this summary is inputted into a random forest classifier that determines the most suitable available position strategy from the summary, generating a list of recommended positioning strategies in order of relevance for the given summary.

Stage 3 is the final operative stage, during which the actual positioning is performed

following the recommendations in order of relevance. If access to peripherals is required, such as when a self-positioning strategies is recommended (e.g., GPS), a positioning request will be sent towards stage 1 through stage 2 to use the corresponding peripheral. Otherwise, collaborative positioning is performed directly using the information from the summary.

4.1.1. CAMPOS Positioning Algorithm

All three stages of the positioning model are encompassed in Algorithm 4-1. The entire process begins when a consumer application requests that the model performs a positioning estimation, represented by the `estimatePosition()` function.

The first step is assembling the device (D) and scenario (F) feature vectors, which is achieved via the `requestContextSensing()` function. For purposes of simplifying the algorithm's explanation, we assume that both of these feature vectors are updated through `requestContextSensing()`, because although their assemblages are made through separate processes, they are always carried out simultaneously.

Algorithm 4-1. Handling a positioning request from a consumer application (C++ pseudo-code)

```
position estimatePosition():
    deviceFeatureVector D
    scenarioFeatureVector F
    [D, F] ← requestContextSensing ()
    scenarioFeatureVector F ← requestContextSensing ()
    if (contextDB.hasPreviousKnownPosition ())
        if (NOT contextDB.hasContextChanged (D))
            return contextDB.lastKnownPosition()
        else
            if (NOT onTimeOut())//not waiting, numberOfAttempts <= n
                scenarioSummary S ← processFeatureVectors(D, F)
                recommendationVector R ← getRecommendations(S)
                return requestPositioning(S, R)
    else
        if (NOT onTimeOut ())
            scenarioSummary S ← processFeatureVectors(D, F)
            recommendationVector R ← getRecommendations(S)
            return performPositioning(S, R)
```

Once the feature vectors have been assembled, the model checks the contextual database stored in its memory to determine whether the device has a valid last known position stored or not (`contextDB.hasPreviousKnownPosition()`). If there is a last known position, the model proceeds to determine whether the device's context has changed or not since that estimation was performed (`hasContextChanged()`). Otherwise, the model proceeds to perform a positioning estimation, a process that is detailed further down.

If the context has not changed (`hasContextChanged() = false`), the model returns the last known position (`contextDB.lastKnownPosition`) to the consumer application, ending the positioning request in success and updating the `contextDB`.

On the other hand, if the context *has* changed, a new positioning estimation must be performed. The `onTimeOut()` function determines whether the model is ready to perform positioning or not. A device is put on timeout when several failed positioning estimations are performed, which allows to save energy by avoiding unnecessary peripheral utilization, and to avoid flooding the network with positioning requests.

If a device is in timeout, the current positioning request immediately ends in failure without triggering a new positioning request. Otherwise, the model processes the scenario feature vector (`processFeatureVectors()`), extracting relevant information and deleting duplicate and irrelevant elements, resulting in the contextual summary S. The model then uses the summary S to determine the recommended positioning strategies vector R via `getRecommendations()`, and request that the actual positioning is executed.

The `performPositioning()` function determines how the positioning is performed, based on the summary S and recommendations R. If the recommendation is to perform collaborative positioning, the model uses the neighbor nodes' information stored in S to perform trilateration. If the recommendation is to access any positioning strategy present in the environment, the positioning is delegated to the corresponding peripherals.

Either way, the positioning attempt can end up in success or failure. In the event of success, the estimated position is stored in the `contextDB` and relied back towards the consumer application. For a failure, there can be two outcomes; either a new positioning request is scheduled, or the device is put on timeout.

4.1.2. Sequence call diagram

Figure 4-2 shows the sequence call diagram of the model's behavior, which helps in understanding how the stages relate to each other and how the different functions are called. The positioning estimation begins with a consumer application requesting the position of a device (e.g., when FourSquare needs to know the position of its user after he/she attempts to log into a restaurant). Stage 2 (context information management) receives this request and asks stage 1 (context sensing) to assess the device's environment and gather contextual information relevant to the positioning process, which is then formatted into a feature vector and sent back towards stage 2.

The sequence of a positioning estimation begins with a consumer application requesting the position of a device (e.g., when FourSquare needs to know the position of its user after he/she attempts to log into a restaurant). Stage 2 (context information

management) receives this request and asks stage 1 (context sensing) to assess the device's environment and gather contextual information relevant to the positioning process, which is then formatted into a feature vector and sent back towards stage 2.

If the device has had a position estimated prior to the current request, and there have been no relevant changes to the device's context (i.e., it has not moved since it acquired its last position, and that position was acquired within a time lapse determined by the consumer application's relevance factor), stage 2 bypasses stage 3 and returns the last known position as the current position.

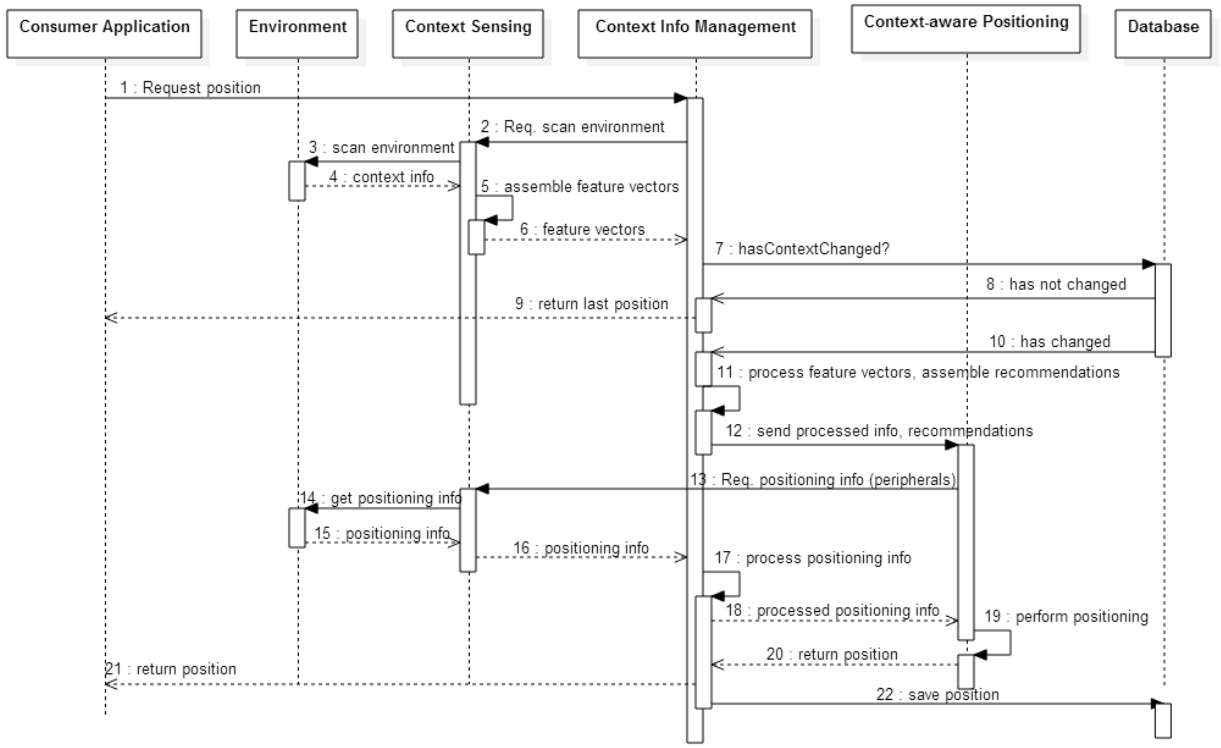


Figure 4-2. UML sequence diagram of the model's behavior

If the context has indeed changed, or the device did not have a last known position to begin with, stage 2 checks whether the device is in "time out" or not. A device is put on time out when a given number of failed consecutive estimation attempts are reached, or a number of successful context-sensing requests yield similar results (i.e. no change, or not enough information to perform an estimation), after which the node is told to wait for a time t. If the device is in timeout, stage 2 will abort the estimation attempt; otherwise it will process the feature vector and assemble the contextual summary, and determine its corresponding recommendations, which are sent towards stage 3.

Finally, during stage 3 the recommendation vector's strategies are read in order of relevance, and the model attempts to execute the corresponding positioning strategy, requesting access to peripherals for self-positioning, or using the summary for collaborative positioning. If the position is successfully estimated, it is sent back towards

stage 2 and stored, and relied back to the consumer application. On the other hand, if the position is not acquired, a new positioning request will be scheduled (or the device will be put on time out, depending on the number of attempts).

In the following sub-sections we describe in detail the context-aware positioning model, as well as the specific actions conducted during each of the stages that comprise it. In addition, Subsection 4.5 offers a thorough example scenario in which the model would prove useful.

4.2. Stage I: Context Sensing

The context sensing stage of the model involves sensing the physical environment of a particular device, in order to determine which contextual elements are present at the time a positioning request is requested, as well as interacting with these elements. In order to achieve this, a device must be able to detect, identify, and characterize contextual elements from both its physical environment and other devices in its vicinity, which can then be used to help describe and infer the target users' location context.

The contextual elements that we have taken into account when designing our positioning model (i.e., what we have defined as our context) include the communication protocols and technologies that the target device possesses, which determines its connectivity and directly affects the amount of neighboring devices that can be reached; the positioning strategies that are available in the environment, and that can be accessed by the device; and neighboring devices, which includes both mobile (e.g., smartphones, tablets, etc.), and stationary elements (e.g., access points, other devices with antennas).

The specific communication protocols considered taken into account are TCP/IP and UDP (transmission control protocol/Internet protocol and user datagram protocol, respectively). The communication technologies are GRPS (general packet radio service), HSDPA (high speed downlink packet access), LTE (long term evolution), Wi-Fi, and Bluetooth. As for the devices that participate in the positioning process, the model does not require any particular type of device to work; as long as the participants can perform self-positioning and/or communicate with other devices, and have enough energy available, they can be part of the positioning process.

The positioning model assumes the existence of an *ad hoc* communication infrastructure in the work scenario (e.g., in the example provided in Subsection 4.5, Freeside, the earthquake-stricken town) that implements unstable communication links among the nodes, e.g., a mobile *ad hoc* network (Figure 4-3).

The model also considers that networking issues related to communication (such as, the discovery of neighboring devices and the management of connections/disconnections) are addressed by the communication protocol that implements these links, which is entirely dependent on the devices capabilities. For

instance, one device might have the Optimized Link State Routing Protocol (OLSR) [76] implemented, while another type of device might have the High Level MANET Protocol (HLMP) [136]. In other words, the positioning model runs into the application layer, and it uses the communication services provided by the networking layer.

Figure 4-3. A device sensing its context

Regarding the positioning strategies, several types are considered, including but not limited to, fingerprinting, RFID, trilateration, and GPS. Again, note that the positioning strategies available for a given device are independent from the model, and rely entirely on the device's capabilities; i.e., if no GPS transceiver is present, the model will not recommend the use of GPS, even when it would be the preferred strategy otherwise. A basic illustration of the process of assembling the feature vectors is shown in Figure 4-4.

The devices' peripherals are also taken into account, since they determine which contextual elements can be sensed and interacted with. When the model requests an interaction with the environment, the associated peripherals convey the required data. For example, when the consumer application requests a positioning estimation, the model will use the device's peripherals to assess the scenario; similarly, if the model requests performing dead-reckoning, the digital compass will return the device's current direction and the accelerometer the movement speed.

Subsection 4.5 provides a case of use of our model. In such example, this stage occurs at the beginning and end of each of the portrayed positioning attempts, specifically during (a) the context sensing performed by each of the example teams' transceivers; (b) when communicating with other devices asking for positioning information for collaboration; and (c) when requesting access to or use of a particular positioning strategy present in the environment. In the sequence diagram shown in Figure 4-2, this stage is represented by the entity called "Context Sensing", which interacts mainly with the entities

“Environment”, which represents the real-world scenario, and “Context Sensing”, which is described in this subsection.

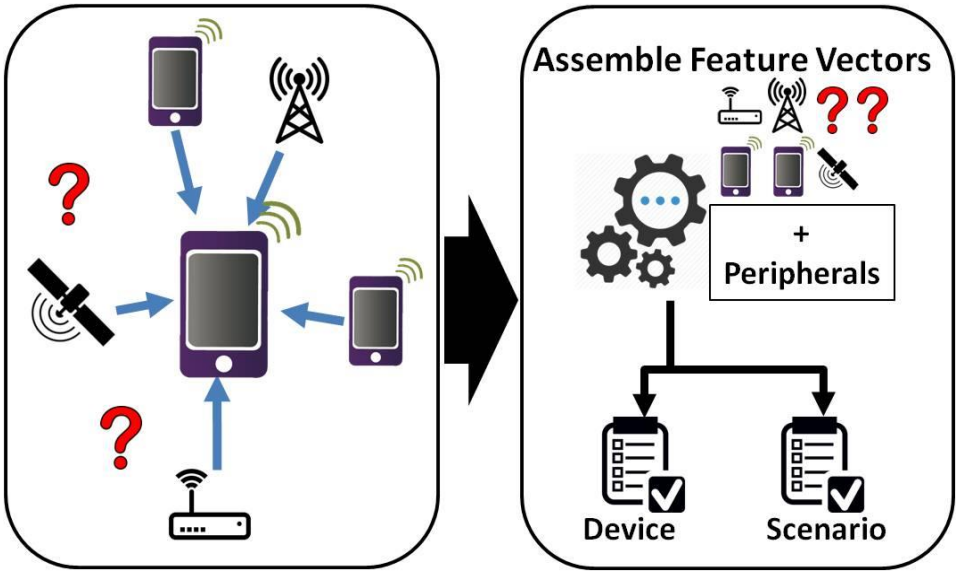


Figure 4-4. Characterization of the environment

During the context sensing stage, a device can perform the following tasks: (1) characterizing the physical environment and (2) interacting with that environment. These tasks are described in more detail in the following subsections.

4.2.1. Characterizing the Physical Environment

In order to allow the model to access and consume information from its device’s context, the latter must be represented in a simple and discrete manner. Thus, we have assembled the relevant contextual elements into two feature vectors; one that describes neighboring devices and another that describes the physical scenario. Each device assembles its own feature vector, and shares it with its neighbors when requested.

4.2.1.1. Device Feature Vector

The device feature vector (Table 4-1) stores the contextual information of a given device, based on its available peripherals and capabilities. It allows for describing devices in terms of both their capability to sense and access the environment, represented by the communication protocols, technologies, and peripherals they possess; and their own fitness to participate in the positioning process, represented by the positioning peripherals they have installed, whether they know their position or not, and a fitness score. This means a device is likely to behave differently depending on which contextual elements are available at a given time.

Table 4-1. Device feature vector.

Communication	List of available communication protocols (i.e., TCP/IP and UDP).

Protocols	
Communication Technologies	List of available technologies (e.g., GRPS, Wi-Fi, etc.).
Positioning Capabilities	List of positioning strategies (e.g., GPS, fingerprinting, etc.) that can be used.
Peripherals	A list of available sensors and actuators, each with their current measurements (e.g., GPS transceiver, Bluetooth, accelerometer, etc.)
Position	A list of a device's last known positions, including their accuracy and positioning decay.
Fitness	An indicator of whether the device is apt to participate in (collaborative) positioning.
Energy	The device's maximum and current energy levels.

4.2.1.1.1. Fitness Indicator

"Fitness" is a dynamic indicator that we have developed, which allows us to determine a device's aptitude to perform positioning. The greater the fitness of a device, the more likely its positioning information could be useful during collaborative positioning. We have defined *fitness* as the product of a *decay* indicator, an *accuracy modifier*, a *position* switch (1 or 0), and an *energy modifier*. Formally, fitness is calculated as follows:

$$\mathrm{F} = \mathrm{D} * \left(\mathrm{A}\frac{\mathrm{A_i} - 1}{\mathrm{A_i}}\right) * \mathrm{P} * \frac{\mathrm{E}}{\mathrm{E_{max}}} \qquad \textbf{(1)}$$

Where F is the fitness score of a device, which can range from 0 to 100 (arbitrary units). D is the decay of the last known position estimation. The parenthesis element of formula 1 represents the *accuracy modifier*, where A is the accuracy requirement of the consumer application, and $\mathrm{A_i}$ the accuracy of the last positioning estimation (both percentages). P is a binary value representing whether the device knows its position or not (i.e., has a last known position). E is the current amount of energy, and E_{max} the maximum energy level of the device, both represented as arbitrary units, and the division of both results in the fitness' energy modifier.

4.2.1.1.2. Decay Indicator

Since the scenario is mobile, and thus highly volatile (i.e., most devices are moving), we have assigned a Decay indicator (D) to the positioning estimations, in the range of 0 to 100, representing the "validity" of these estimations as time passes and the devices move. Decay is represented by the following formula:

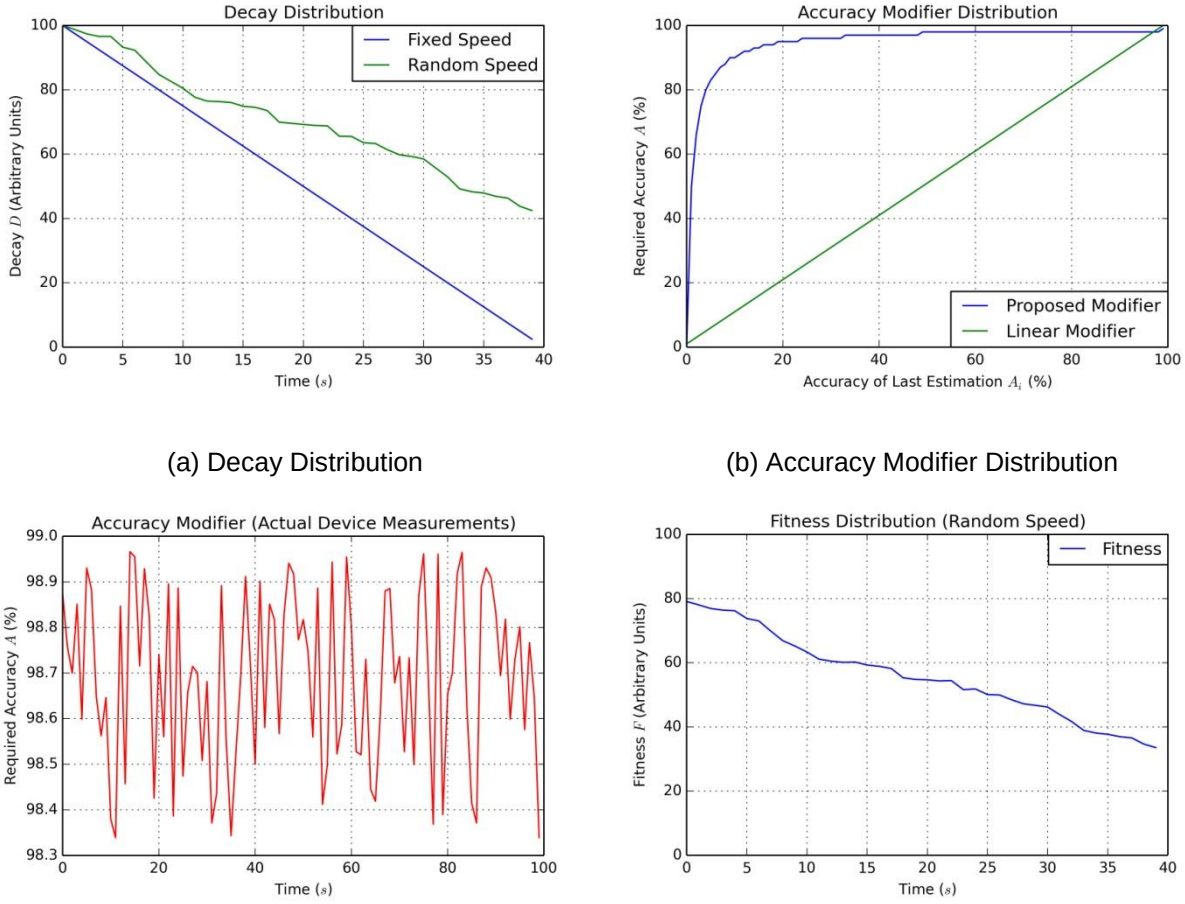
$$\mathrm{D(s,t)} = 100 - \mathrm{k} * \sum_{\mathrm{i=1}}^{\mathrm{t}} \mathrm{s_i}, \qquad \textbf{(2)}$$

Where D is the decay score, s is the speed of the device, t represents the time from the last positioning estimation, and k is a coefficient modifier, determined based on the type of node (for more on node types, refer to Subsection 5.1).

The decay indicator (D) of a device is initialized at a value of 100 whenever it

estimates its position successfully, and then it decreases steadily based on the user's speed on the field. Once the decay reaches a threshold of 60, the model will begin requesting new positioning estimations, because it is assumed that the current position is no longer representative of a device's true location. If the decay keep decreasing until reaches a threshold of 20, the node's position is automatically set to unknown ($P = 0$).

Figure 4-5-a shows the distribution of the decay of a positioning estimation for a single device over time. The blue line shows a stable decrease, which would only occur if the device has a fixed speed throughout the whole process; while the green line shows a staggering decrease, more similar to real world conditions in which the speed of the user's varies over time. For more on decay, please refer to Subsection 5.1.



(a) Decay Distribution

(b) Accuracy Modifier Distribution

(c) Accuracy Modifier Example

(d) Fitness Distribution

Figure 4-5. Fitness, Decay, and Accuracy Modifier distributions

The accuracy modifier is a factor that allows us to apply a penalty to the fitness based on how far the accuracy of the last known position ($\mathrm{A_i}$) is from the accuracy required by the consumer application (A). If the device has no last known position, the value of $\mathrm{A_i}$ is set at 1 to make the accuracy modifier equal to zero, which also makes fitness equal to

zero; otherwise, the value of $\mathrm{A_i}$ is set to the accuracy percentage of the last known position.

Figure 4-5-b shows two distributions of the accuracy modifier. Our distribution is represented by the blue curve, which is close to a logarithmic scale, and allows us to apply only a small penalty to the fitness; whereas the green curve shows a linear distribution, which applies a huge range of possible penalties, depending on the $\mathrm{A_i}$ values. For instance, using our distribution, the accuracy modifier for $\mathrm{A_i}$ values of 40% and 60% with a requirement of $\mathrm{A} = 100\%$ is roughly the same, 97.3% and 98.6%, respectively. Conversely, the linear accuracy modifier would yield modifiers of 40% and 60% for the same A and $\mathrm{A_i}$ values, far more punishing to the fitness score. Figure 4-5-c shows an example of the accuracy modifier values of our distribution through time.

The energy modifier denoted by E and $\mathrm{E_{max}}$ works in a similar way to the accuracy modifier, but is a linear distribution, i.e., literally the percentage of energy the device has available. We opted for a linear distribution instead of a smoothed one because energy is an ever present concern for mobile devices. The lower the energy levels, the more punishing the modifier to fitness, and the less likely the device will be forced to participate in collaboration when its energy level is low.

Finally, the P indicator works as a switch that allows the model to override any previous fitness value, effectively turning it into zero and forcing the device to perform a new positioning estimation. This is useful when the accuracy requirement of a consumer application varies over time, or when the user directly requests a positioning estimation, since it is much faster than waiting for the fitness score to decrease naturally.

In general, devices with greater decay (i.e., lower D values) and/or lower energy values would tend to have low fitness scores; on the other hand, devices with lower decay (i.e., greater D values) and/or energy values yield higher fitness scores. Devices that do not know their position would have a fitness of zero ($if\ P = 0\ or\ A_i = 0, then\ F = 0$), since they would not contribute useful information during the positioning process or have too low energy levels to risk participation.

Figure 4-5-d shows an example of the Fitness curve of a device. Note that the fitness is very similar to the decay curve shown in Figure 4-5-a; this is because fitness is strictly dependent on decay, but is adjusted by the accuracy and energy modifiers to reflect the state of a device during the positioning process. Thus, where decay dictates when a positioning estimation must be performed, fitness determines whether a device should attempt or participate in such a process.

4.2.1.2. Scenario Feature Vector

The scenario feature vector assembled by our model represents the immediate context-scenario of the device. This scenario feature vector brings together information from all accessible (and available) contextual elements in the devices' proximity, allowing

a device to access that information afterwards, as required.

In addition to the contextual elements from the surrounding environment of a device, this characterization also holds a list of the positioning strategies that can be accessed by such a device, as shown in Table 4-2.

A given device possesses only one scenario feature vector at a time, which is updated each time it performs a new context sensing. This is done for ease of access and storage during later stages of the positioning process.

Table 4-2. Scenario feature vector.

Access Points	List of access points available in a device's context.
Neighbors	List of neighboring nodes available in a device's context.
Positioning Strategies	List of positioning strategies available in a device's context.

A device would update its scenario feature vector each time it attempts to perform positioning, effectively assembling a map of the assets in its surrounding area; detecting access points, available positioning strategies, and asking for its neighbors' device feature vectors. Neighbors that are fit to participate in collaboration would send back their own device feature vector, while those that are not able simply would not take any actions.

Although access points are not "devices" in the sense we have defined in this thesis, they are also represented by using a device feature vector assembled on the fly, which only saves information related to whether the access points know their position or not, and it is assumed that they have infinite energy and a 100 fitness score. This way, access points are treated as stationary devices, and these are always preferred over mobile neighbors when performing collaborative trilateration, as long as they have known positions.

Algorithm 4-2 shows a simplified version of the `requestContextSensing()` component of the model, which is in charge of performing the scenario-sensing and the assemblage of the feature vectors for the device (D) and scenario (F). To this end, each supported peripheral must be activated individually, in order to determine the availability of potential collaborators and positioning strategies, and all the information collected stored in the device and scenario feature vectors.

The main idea of the algorithm is that for each supported peripheral, a discover signal or "ping" is broadcast to the environment, and if a response is received, an action is taken depending on the type of peripheral.

For visualization purposes, instead of showing the activation of each peripheral, we have grouped them into three categories; communication, navigation, and positioning. For the `communicationPeripheral` category, if the device possesses the peripheral in question, it is added to the feature vector D.

In addition, any device that responds to the ping is added to the `neighborList`, which stores all neighboring devices and access points, independent of their fitness, which is in turn added to F. In the case of the `navigationPeripheral` category, if the device possesses the peripheral, it is immediately added to D. Finally, for the `positioningPeripheral` category, if the device possesses a positioning capability transceiver (or algorithm), it is automatically added to feature vector D, and if its corresponding positioning strategy is available in the scenario (`P.ping == true`), it also added to F.

Algorithm 4-2. Assembling the device and scenario feature vectors (C++ pseudo-code)

```
scenarioFeatureVector requestContextSensing():
    deviceFeatureVector D
    scenarioFeatureVector F
    for each communicationPeripheral C in device
        D.add(C)
        neighborList L ← C.ping()
        F.addNeighbors(L)
    for each navigationPeripheral N in device
        D.add(N)
    for each positioningPeripheral P in device
        D.add(P)
        if (P.ping)//if strategy P is available in scenario
            F.add(P)
    return [D, F]
```

After a device has finished scanning its environment, it will populate its scenario feature vector with the pertaining information, storing it in an Extensible Markup Language (XML) file. We decided to use an XML representation not only because it is a standard format, but also because it is flexible and allows for information integration using well-known mechanisms.

Thus, several feature vectors can be integrated into a large one, involving low computing effort. Given the diversity of devices participating in outdoor activities, counting on a common representation of the shared information is mandatory to ensure data interoperability. In this sense, the use of XML to represent the shared information seems to be the best option. An extract of the summary of a device is shown in XML format in Algorithm 4-3.

Once the scenario feature vector of a device has been completed, the assorted XML file is sent towards the management of contextual information (presented in Subsection 4.3) for processing.

Algorithm 4-3. Extract of the representation of the device and scenario feature vectors (XML)

```
<?xml version="1.0" encoding="utf-8"?>
    <data>
            <device>
                    <commProtocols>
                            <protocol>0</protocol>
                    </commProtocols>
                    <commTechnnologies>
                            <technology>1</technology>
                            <technology>3</technology>
                            <technology>4</technology>
                    </commTechnnologies>
                    <positioningCapabilities>
                            <posStrategy>0</posStrategy>
                            <posStrategy>5</posStrategy>
                            <posStrategy>6</posStrategy>
                            <posStrategy>7</posStrategy>
                    </positioningCapabilities>
                    <peripherals>
                            <peripheral></peripheral>
                    </peripherals>
                    <position>
                            <latitude>40.748440</latitude>
                            <longitude>-73.984559</longitude>
                    </position>
                    <fitness>67</fitness>
                    <energy>2445</energy>
            </device>
            <scenario>
                    <neighbors>//includes devices and Access points
                            <device>...</device>
                            <device>...</device>
                            ...
                    </neighbors>
                    <positioningStrategies>
                            <posStrategy>0</posStrategy>
                            ...
                    </positioningStrategies>
            </scenario>
    </data>
```

4.2.2. Interacting with the Physical Environment

This task deals with the use of a device's communication and navigation peripherals to interact with the environment. It is usually requested during the context sensing and context-aware positioning stages, as a means to detect surrounding assets, to use specific positioning strategies, or to communicate with neighboring devices. Figure 4-6 shows an interaction between a device and GPS satellites, after which it obtains its position, and stores it for later use.

The considered communication peripherals include all types used to send or receive

information, including Global System for Mobile Communications (GSM), Wi-Fi antennas, RFID card readers, and GPS transceivers. These are used to send and receive information among neighbors and access points, and indicate which devices can communicate with each other, since only devices that share communication channels can "talk". For example, two devices with Wi-Fi capabilities can safely exchange information through that channel; however, any data they send will not be visible to devices without Wi-Fi, even if these devices are able to communicate with others through other channels such as GSM.

Figure 4-6. Interacting with the physical environment

As for navigation peripherals, they include the magnetometer or digital compass, used to determine the general direction of the device; the accelerometer, used to detect movement and measure traveled distances; and a gyroscope, used to determine the relative position of the device with respect to the plane. These peripherals are fairly common in most modern smartphones, but older devices may not possess them. Their main use is to provide a device with a degree of relative positioning, by using dead-reckoning (discussed further at the end of this Chapter) when no other positioning strategies are available and a device knows its last position.

Any information gathered during this task is sent back towards the requesting entity, generally the context sensing component during the sensing of the physical scenario (presented in Subsection 4.2.1), or during the execution of the context-aware positioning component (explained in Subsection 4.4).

4.3. Stage II: Management of Contextual Information

This is the second stage of the model, and the key part of the context-aware positioning process provided by it. During this phase, the contextual elements from a device's context

are assessed and processed in order to eliminate redundant and irrelevant information related to the positioning of a device.

When a consumer application requests a positioning estimation, the contextual information manager sends a request to the context sensing. Based on the information gathered from the device's surroundings, it constructs a set of positioning strategies recommendations. These recommendations consist on a list of the positioning strategies that can be accessed by the device, prioritized according to the device's current context and expected energy consumption level. The reference values for this last feature were obtained from several studies reported in the literature.

In order to assemble the recommendation set, the considered positioning strategies are treated as classes and represented as a vector. Using the contextual information previously gathered, a set of randomized decision trees are used to assign scores to each positioning strategy in the vector, representing the likelihood of a device belonging to a class (i.e., to a positioning strategy). The higher this score, the greater the estimated suitability of the associated positioning strategy given the device's context.

Three tasks can be performed during this stage: (1) request sensing the physical environment; (2) process feature vectors and assemble recommendations; and (3) manage positioning information. These tasks are described in detail in the following subsections.

4.3.1. Request Sensing the Physical Environment

This task is triggered when the consumer application of a device requires a positioning estimation to be executed, or when a device's decay indicator of a known position has crossed the established threshold and thus is no longer reliable.

The actions of this task are very straightforward. The contextual information manager sends a context-sensing request to scan the environment and assemble the device and scenario feature vectors. The task is completed when the context sensing component sends back a raw XML file containing the feature vectors described in Subsection 4.2, which are then directed towards the next task for processing.

4.3.2. Process Context Feature Vectors and Assemble Recommendations

During this task, the scenario feature vector is processed, in order to extract relevant, ignore irrelevant, and update redundant information, by means of low-level data fusion [64]. The model uses the raw contextual information obtained during the context sensing stage, along with any other data stored from previous positioning attempts, in order to produce an up-to-date representation of the scenario. Next we explain the process followed by CAMPOS to obtain the recommendation vector.

4.3.2.1. Data Fusion

The data fusion is a straightforward process, which varies depending on which contextual elements are being assessed. Since most of the devices' features are static, the data fusion concerning peripherals, communication protocols, technologies, available positioning strategies, and energy levels, is simply an update over the last known data that occurs only when changes are detected.

This is because it is highly unlikely that a user would physically add or remove additional hardware elements while on the field (e.g., adding a GPRS antenna), but it is still flexible enough to allow these elements to change through time (e.g., energy levels), or to be unavailable due to external conditions (e.g., a RFID card reader becomes unable to read nearby tags, or a GPRS base station remains temporarily unreachable).

When a new scenario feature vector is processed, the first step is to prune neighbors that are not useful for collaborative positioning. To this end, the model automatically discards all nodes with fitness scores below a threshold of 60 (on a scale of 1–100), and the remainder are used as input. Note that this threshold is dynamic, and it can be lowered by the model in subsequent positioning attempts if it is unable to find enough candidates using that threshold.

For example, if the model chooses collaboration as the only possible positioning option, and no nodes are found at a fitness threshold of 60, the next positioning attempt of the model would be performed at 55; if only one node is found at that level, the next attempt will be made at a threshold of 50, and so on, until an adequate amount of nodes is found.

Conversely, if the average fitness of the pruned nodes is over the current threshold, the threshold for the next positioning attempt will increase accordingly. If the threshold ever decreases to 20, the node's position is automatically set to unknown, and the threshold is reset to its initial value of 60.

The model then compares and updates any information changes between the recent contextual and stored data. The structure of the XML feature vectors allows for a rapid comparison of discrepancies between a node's current and previous scenario information, facilitating this process. If any discrepancy is found, it is immediately updated.

For instance, if we have information from node N_7 in storage, and receive new information from context sensing (N_7'), the model will probably detect changes only in the positioning capabilities, energy levels, and position features, since they change over time. On the other hand, the communication protocols, communication technologies, and peripherals features are likely to remain the same, unless direct action from the user is taken to shut them off, in which case the data is updated.

The remaining nodes are then ordered based on their fitness scores, for easing the

access. This resulting subset should contain only viable candidate nodes for collaboration, essentially a summary of the collaboration work context of a device. This summary is in fact a trimmed version of the original scenario feature vector, which only contains the positioning information of viable neighbors, as well as the positioning strategies available in the target device's current context.

Algorithm 4-4 details the processing of the device and scenario feature vectors to construct the summary S. As explained above, the first step is updating the status of the stored device feature vector (`contextDB.deviceFeatureVector`) to match the current estimation's features from feature vector D.

Then, the positioning strategies from the scenario feature vector F are assigned to the summary, and the devices are pruned based on the fitness threshold relative to the current positioning estimation (which is determined based on the amount of estimation attempts that have been performed).

Algorithm 4-4. Processing the device and scenario feature vectors (C++ pseudo-code)

```
scenarioSummary processFeatureVectors(deviceFeatureVector D,
                                      scenarioFeatureVector F):
    scenarioSummary S
    if (contextDB.deviceContextHasChanged(D))
        contextDB.updateDeviceFeatureVector(D)
    S.deviceFeatureVector ← contextDB.deviceFeatureVector
    S.scenarioFeatureVector.posStrategies ← F.posStrategies
    S.scenarioFeatureVector.neighborlist ← getFitnessPrunedDeviceList(F)
    return S
```

4.3.2.2. Assembling the Recommendation Vector

Once the data fusion is finished, the next step involves assembling the recommendations using the information from the summary. To this end, the positioning strategies are treated as classes, and represented as a vector (Table 4-3). Each strategy in the vector has an associated score, which represents the likelihood of a device belonging to that class (i.e., positioning strategy), given its contextual information.

The higher this score, the greater the estimated suitability of the associated positioning strategy. All scores must add up to 1.0, regardless of the number of positioning strategies observed by the model.

A total of eight positioning strategies have been considered when designing CAMPOS. Five of these strategies are dependent on the scenario and the devices' positioning capabilities, i.e., self-positioning strategies: GPS, A-GPS, Cell-ID, fingerprinting, and RFID (S1 through S5).

Table 4-3. Example of a recommendation vector.

Strategies	S1	S2	S3	S4	S5	C1	C2	DR
Score	0.35	0.15	0.05	0.10	0.30	0.0	0.00	0.05

Two additional strategies can be estimated directly by the model by using information from the summary, i.e., collaborative positioning strategies: triangulation, and minimum bounding rectangle (C1 and C2, respectively).

Finally, dead-reckoning (DR) is a self-positioning strategy that uses input from non-positioning-specific peripherals as well as stored positioning information to guess the current position of a device.

Any number of additional positioning strategies can be added to our model without restriction, but retraining is necessary for the model to consider them during the classification process.

In order to build the set of positioning recommendations, the model attempts to determine the suitability of the positioning strategies, by using a randomized decision tree. Thus, each time the model attempts to classify a device (i.e., to determine its best positioning strategy choice), it picks a "remaining" attribute from the device's associated scenario feature vector at each node expansion.

A categorical feature (e.g., positioning capabilities) is considered "remaining" if the same categorical feature has not been chosen previously in a particular decision path, starting from the root of tree to the current node. However, continuous features (such as fitness or current energy level) can be chosen more than once in the same decision path, and each time that such a feature is chosen, the threshold is determined randomly.

Depending on which decision path is chosen in each decision node, the scores of the recommendation vector's positioning strategies are modified. Once all the attributes from the device's feature vector for such a scenario have been chosen (i.e., there are no remaining features), the recommendation vector is considered to be assembled.

Figure 4-7 shows an example of the process of assembling a recommendation vector using a set of randomized decision trees. X represents the input of the tree, i.e., the device's associated scenario feature vector. φ_1 and φ_n represent two different random decision trees, and $\varphi_i = (Y = c|X = x)$ represents the probability of obtaining the recommendation set Y given feature vector x. A total of ten randomized trees are used to assemble the recommendation vector, by averaging their individual scores.

Figure 4-8 shows an example of a *random* decision tree, fully grown. If we were to use this type of decision tree, our decision process would be heavily dependent on chance, since once a condition is met (or not met), we would automatically be unable to access any other cut condition that is a child of the not chosen branch. For instance, if the device has more than 63%, it would follow the right branch of the tree from Figure 4-8, entirely missing the left branch fingerprinting, GPS, and Cell ID positioning strategies, even if the device is capable of accessing those strategies.

Moreover, some features are closely related, such as having an accelerometer, digital compass, gyroscope, and a last known position, all of which lead towards dead-reckoning. In a random tree these features can appear in any order, and if one of them is not met then dead-reckoning would be ruled out immediately.

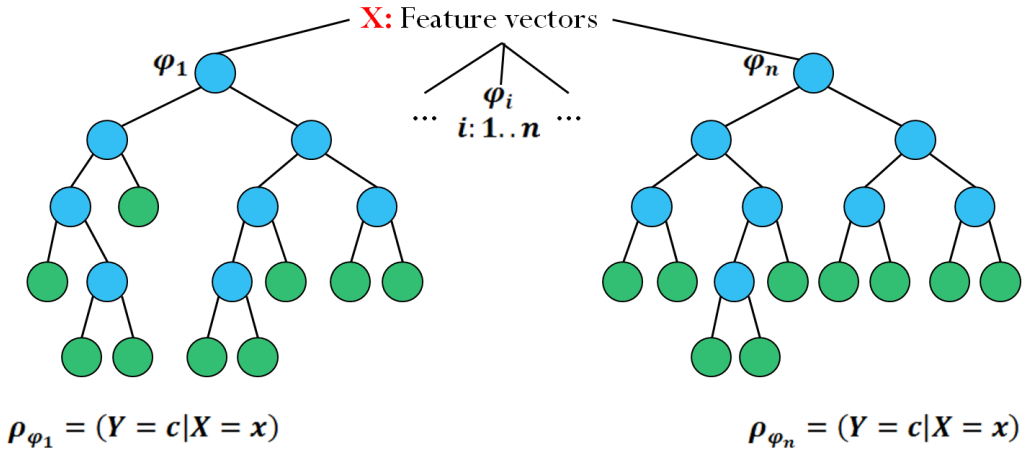


Figure 4-7. The model's recommendation assembling.

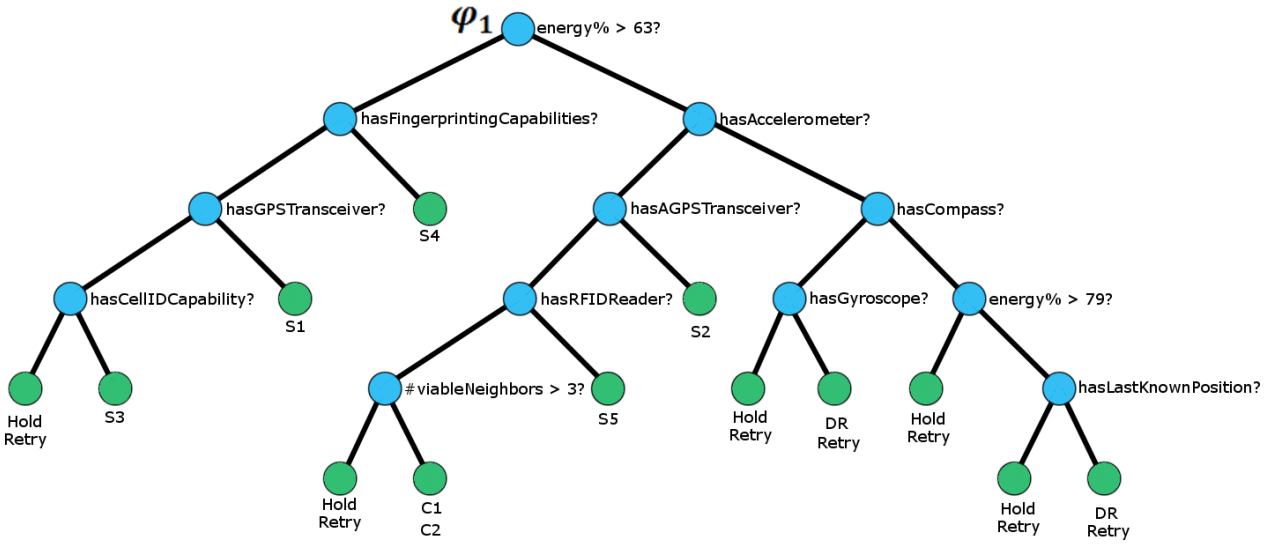


Figure 4-8. Example of a random decision tree

Thus, we use *randomized* decision trees, in which the tree grows one node at a time, randomly choosing the current condition from the pool of available features, whether the past condition was met or not, until a positioning strategy (i.e., a leaf node) is chosen. Given that the cut conditions of the branch nodes are selected at random (using the guidelines explained earlier in this subsection), the order in which these conditions are chosen is decisive when determining the recommendation vector scores of a specific tree.

This means that a device possessing a GPS transceiver, a fingerprinting app, and enough energy available, would immediately prefer whichever positioning option that appeared first, regardless of its contextual suitability. Thus the need to use averaging. By

letting several trees (P_{φ_i}) cast their votes, we ensure variety in the final recommendation vector over the course of several attempts (i.e., there will be more than one option), while statistically allowing the better ranked positioning strategies to remain as the preferred options. An example of a randomized tree is shown in Figure 4-9.

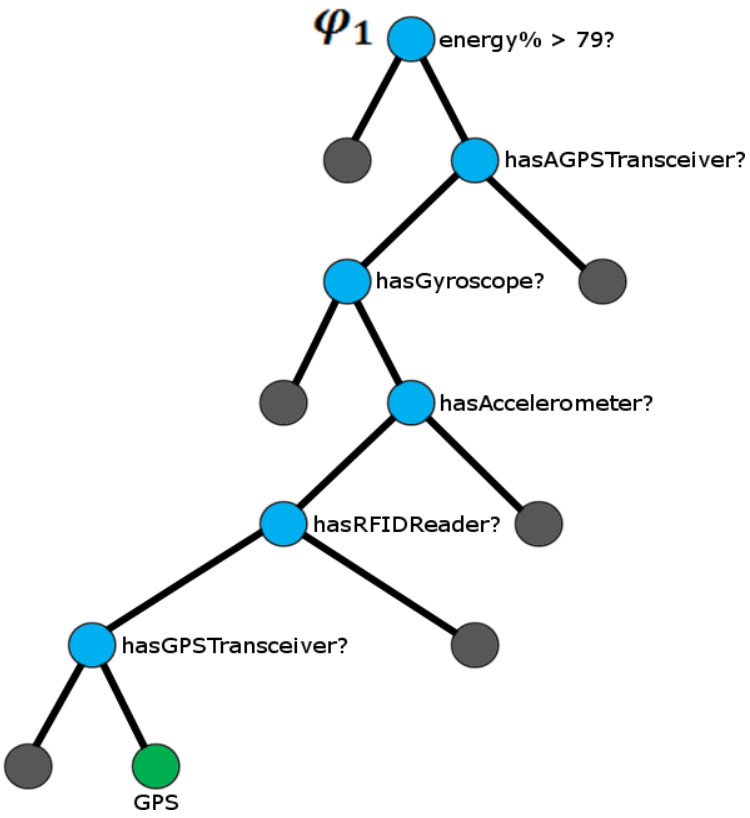


Figure 4-9. Example of the randomized decision trees used by our model

Table 4-4 shows a few examples of scenario feature vectors, each representing the sensed context of a device at a given time. These vectors were used as training data for the model, in order to tailor the thresholds of the decision tree nodes. Note that all of the scenarios show the collaborative and dead-reckoning positioning strategies (C1, C2, and DR, respectively) as available, independent of other conditions.

This does not necessarily imply that these strategies can readily be used, only that they are potentially available to the owner of the vector. To illustrate this fact, let us think of a bus station; although a certain bus line is known to pass through the station, there is no guarantee that a bus from that specific line will be waiting for you exactly when you need it.

Table 4-4. Example of a scenario feature vector training data.

	Detected Access Points*	**Detected Neighbors**	**Available Pos. Strategies**
Scenario 1	[Dev 1, Dev 2]	[Dev 3, Dev 4, Dev 5]	[S1, S2, S3, S4, S5, DR]
Scenario 2	[Dev 1]	[Dev 3]	[C1, C2, DR]

Scenario 3	[ ]	[Dev 4, Dev 5, Dev 6, Dev 7]	[S1, C1, C2, DR]
Scenario 4	[Dev 1, Dev 2]	[Dev 3, Dev 5]	[S1, S4, C1, C2, DR]
Scenario 5	[Dev 1, Dev 2, Dev 8]	[ ]	[C1, C2]

* Access points are represented as devices by the model.

Each example scenario feature vector has an associated training recommendation vector tailored to its context. These vectors have been assigned scores for each of the positioning strategies supported by the model. These scores were determined based on both results from controlled real world experimental and from literature, and represent the suitability of a strategy given a particular context.

Table 4-5 shown examples of training recommendation vectors, which correspond to the scenario vectors from Table 4-4. Thus, given the *Scenario 3* and *Recommend 3* vectors, the model knows that the most suitable option for positioning given Scenario 3's context is S1 (i.e. GPS), followed by DR (i.e. dead reckoning); and will likely assemble those recommendations for similar scenarios. A score of zero means that a particular strategy is unavailable, due to being inaccessible to the device, or unreachable for any reason.

Table 4-5. Example of a recommendation vector training data.

	S1	**S2**	**S3**	**S4**	**S5**	**C1**	**C2**	**DR**
Recommend 1	0.35	0.15	0.05	0.10	0.30	0.00	0.00	0.05
Recommend 2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
Recommend 3	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.20
Recommend 4	0.70	0.00	0.00	0.10	0.00	0.15	0.05	0.00
Recommend 5	0.00	0.00	0.00	0.00	0.00	0.55	0.45	0.00

Finally, Table 4-6 shows examples of training device feature vectors, representing both access points and neighboring devices portrayed in Table 4-4. The layout of the vector is simple: if a device possesses a given element, then it is assigned a value of one; otherwise, it is assigned a zero. Special values, such as energy and fitness, are represented in arbitrary units.

For example, the device *D3* (shown in Table 4-6) has access to both TCP/IP and UDP protocols, and also to Wi-Fi and Bluetooth. Moreover, it has a GPS transceiver and RFID card reader, a digital compass, and an accelerometer. Additionally, it knows its position (the actual accuracy is irrelevant at this stage, and it is to some extent addressed by the fitness), a fitness score of 56, a battery capacity of 2550 mAh with 3.85 V (equivalent to 9817.5 mWh), and 87% battery left (i.e., current energy pool is roughly 8541.23 mWh). A basic analysis of the energy consumption of mobile device (particularly smartphones) attempting communication with neighboring devices is presented in [111]. For more information on energy consumption, please refer to Subsection 5.1.

Another example to consider is *D1*, which represents an access point. Its device

feature vector shows that it has no positioning strategies available, and yet it knows its position. This is due to the fact that most router and similar access points have embedded technologies that allow them to estimate their positions using IP address location systems, which are based on algorithms that determine the location of an IP address by examining characteristics of the traffic coming from the address, or by looking at the addresses' associated data [45].

Table 4-6. Device feature vector training data example.

	Pro-tocol		Technologies					Positioning Strategies								Peripheral			Knows Position	Fitness	Energy	
	TCP/IP	UDP	GRPS	HSDPA	LTE	WiFi	BT	S1	S2	S3	S4	S5	C1	C2	DR	Accelerometer	Compass	Gyroscope			Current	Max
D1	1	1	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1	100	10	10
D3	1	1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	1	0	1	56	8.5	9.8
D4	1	1	1	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	79	7.6	9.7
D5	0	1	1	1	1	0	1	1	0	0	1	0	1	1	0	0	1	0	0	0	9.3	9.8
D6	1	0	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	2.7	3.6

* Positioning and communication peripherals not featured, since they are part of their assorted strategy and protocol or technology.

All of the training data is inputted and processed during a single training stage that is unique to a given set of positioning strategies. The training determines the values of the thresholds for each of the feature vector's attributes, and their effect over the recommendation vector's strategy scores. After finishing its training phase, the model is capable of choosing the most suitable positioning strategy available to a device, given its current context. As we mentioned earlier, adding further strategies requires retraining of the model to work.

Once the summary from the data fusion phase and the recommendation vector have been assembled, they are stored locally on the target device, for future reference. Thus, the next time the consumer application sends a positioning request to the model, the context information manager component will assemble a new summary and compare it with the latest stored summary, in order to check for changes in the navigation context of the device, i.e., whether the device has moved.

If the model determines that the navigation context has not changed (i.e., the user has not moved), the manager skips the context-aware positioning stage, and just sends back the last estimated position (if available) to the consumer application, and updates the stored data with the newly obtained where required. Otherwise, the summary and recommendation vector are sent towards the context-aware positioning component, and the positioning estimation is performed as usual.

4.3.3. Manage Positioning Information

This task is the final phase of the model, and it is triggered as a result of a successful positioning estimation made by the context-aware positioning component. The actions performed during this task are limited only to directing the device's estimated position towards the consumer application, and also storing it for later use.

Once the context-aware positioning component has acquired a position, either though collaboration or on its own, the resulting data is redirected towards the contextual information management component for the final phase of the positioning process. The positioning data is then stored in the model's database, and redirected towards the consumer application.

4.4. Stage III: Context-Aware Positioning

This is the final stage of the model, and the one in charge of performing the actual positioning estimation. The context-aware component uses the information stored in the summary and the recommendations assembled by the contextual manager component in order to either request the use of positioning peripherals for self-positioning, or use the information from the summary to calculate the target's position through collaboration.

If the recommendation is to perform self-positioning, i.e., to access any of the positioning strategies embedded in the environment (strategies S1 through S5), then the context information manager would receive a request to access the corresponding peripherals (e.g., a GPS transceiver), in order to determine the position of the device. Note that the model only determines which strategy is to be used, and it has no control over the outcome of the estimation. Whether the peripherals actually manage to obtain the desired positioning estimation or not is entirely dependent on the device's capabilities.

On the other hand, if the recommendation is to perform collaborative positioning, the context-aware positioning component would take the information from the summary and the recommendation vector, and then use trilateration to determine the position of the target device. Unlike self-positioning, this process is entirely dependent on the model, because the context-aware component does all the calculations based on the stored contextual information from the summary. No additional peripheral utilization is required, unless too much time has passed since the last reading; in that case, a new context-sensing must be made, and the while process starts over.

When required, the model will attempt to perform positioning using the recommended candidate strategies, according to the contextual information and priorities included in the summary. Two tasks are performed during this stage: (1) request self-positioning, and (2) perform collaborative positioning. These tasks are explained in the next subsections.

4.4.1. Request Self-Positioning

If a device using the model has positioning capabilities of its own, it is very likely that the best rated recommendation will be a self-positioning strategy, i.e., any of the strategies available in the physical scenario. In such a case, the context-aware component would send a request to use the respective positioning peripherals, and expect the resulting positioning information. For example, devices with GPS transceivers or RFID readers could activate them, while Android-based smartphones could use their embedded Wi-Fi positioning system or A-GPS to estimate their positions.

Dead-reckoning (Figure 4-10), also known as deduced reckoning or DR, a special type of self-positioning, is the process of calculating one's current position by using a previously determined position, or by fixing and advancing that position based upon known or estimated speeds, over elapsed time and course [10].

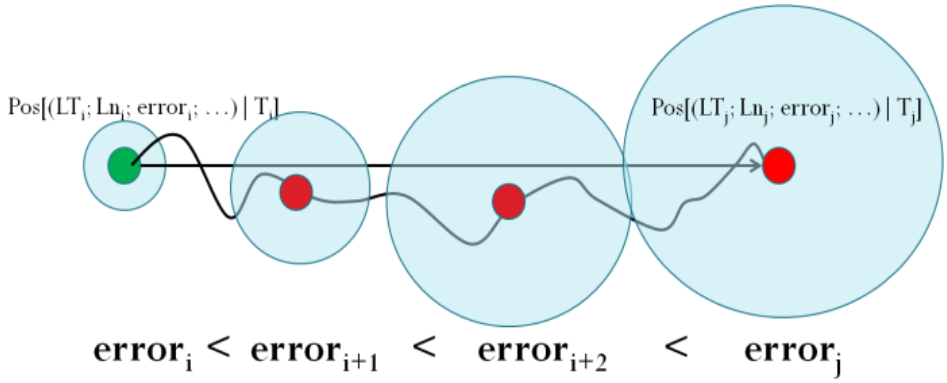


Figure 4-10. Accuracy degradation of Dead-reckoning over time

In the context of our model, dead-reckoning is applied using the list of known positions, in conjunction with the navigation peripherals, in order to smooth transitions between different positioning strategies, since there could be a "blind" positioning elapsed time between using two different strategies, or between two consecutive estimations of the same strategy.

However, DR is subject to cumulative errors, and requires means to correct its estimations by using a proper positioning strategy [10]. It is by no means reliable for use over long periods, and it is relied on only to offer positioning consistency during "blind gaps", when the device does not have access to a proper positioning strategy. The proposed model strictly ignores all nodes using dead-reckoning during the collaborative positioning process.

It is important to note that the actual self-positioning conducted by a device is transparent to our model, i.e., the model recommends using a positioning strategy, but it is up to the device's peripherals to actually acquire the desired position estimation.

Provided all goes well, the device's position is obtained and sent towards the

contextual information management component for processing. In the event of a time-out (i.e., the strategy controller was reached, but no positioning estimation could be obtained), a new request is sent by the model. If this second request also fails, the model requests the estimation to the peripherals associated to the second most suitable strategy from the recommendations vector, and so on. If a total time of 30 seconds passes and no positioning estimation has been obtained, or if the device has run out of available strategies (i.e., it has tried to contact all of them unsuccessfully), the positioning estimation process is halted, and the model puts all of the device's positioning requests on hold for up to one minute while waiting for the context to change.

4.4.2. Request Collaborative Positioning

If the recommendation is to perform collaborative positioning, the model will use the contextual information of the summary to select viable neighboring devices with known positions as reference points. Then, it proceeds to estimate the device's position using either tri-lateration, or minimum bounding box approximation (MBB) [9].

Lateration, discussed broadly in Subsection 2.1, is a type of triangulation in which the position of a resource is estimated by measuring its distance to several reference points with known positions. Using the direction or length of the vector drawn between the location to be estimated and the reference points, it is possible to calculate the absolute position of the desired resource [162]. At least three reference points are required to perform tri-lateration, although more points could be used to improve the accuracy of the estimation, if available (multi-lateration). An advantage of this method is that it involves a small setup effort [69].

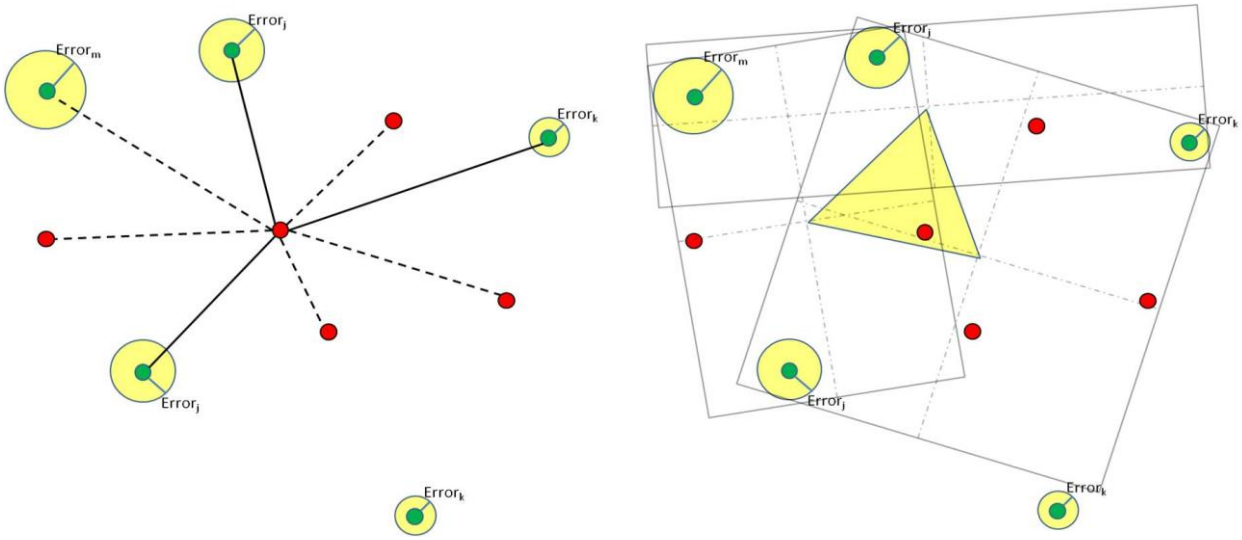
Figure 4-11-a shows an example of tri-lateration performed by the model during a simulation. The green nodes know their positions with fitness scores over the threshold of 60; red nodes are those that do not know their positions, or that have a low fitness score and therefore cannot collaborate with the target. The solid lines link the target to the nodes chosen as reference points, while the dotted lines link nodes that have been acknowledged, but are not part of the collaboration process.

The smallest or minimum bounding box is, for a point set (S) in N dimensions, the box with the smallest measure (area, volume, or hypervolume in higher dimensions) within which all specified points of S lay. The MBB of a point set is the same as the MBB of its convex hull, a fact that may be used heuristically to speed up computation [9]. The term "box" comes from its usage in the Cartesian coordinate system, where it is indeed visualized as a rectangle in two dimensions (2D), a rectangular parallelepiped in three dimensions (3D), *etc*.

Figure 4-11-b illustrates an example of MBB applied to 2D positioning. Using as many nodes that know their positions as possible, a series of rectangles is formed, each containing at least three nodes. Then, line segments are traced from the centers of these

rectangles to each other, forming an irregular polygon (in this case, a triangle), which represents the area where the node is likely located. In some cases, this polygon provides better accuracy than using tri-lateration. However, the cost of calculating this polygon increases in direct relation with the number of rectangles formed.

Although this strategy was tested during early stages of the model, it was put on hold due to the vast additional research effort required to apply it to positioning without increasing the energy and computation requirements. The complexity of the MBB algorithm is of $O(n \log n)$ for 2D and $O(n^3)$ for 3D, with the added requirement of finding the rectangle intersection elevating it further to $O(n^2 \log n)$ for 2D.



(**a**) Trilateration (**b**) Minimum Bounding Rectangle

Figure 4-11. Collaborative positioning strategy options

In the following Chapter, we discuss the experiments performed during the development of the proposed model in depth, describing the simulated scenarios as well as their components, and providing technical insights on how the simulations were constructed and tailored to emulate a real world environment. Using these simulations, we attempt to answer the research questions stated in Section 1.2.

4.5. Using the Model in a Specific Scenario

In order to help the reader understand the sequence of events that occur when the devices use the model, as well as the interactions between devices and environment, we have assembled an example of a disaster-relief scenario, in which different devices attempt to perform positioning while using the model under different conditions and restrictions.

To this end, we have catalogued devices into two roles based on the positioning capabilities they possess. These devices can either be beacons, or non-beacons.

Beacons can access at least one positioning strategy present in the scenario (e.g., GPS), and thus they can perform self-positioning. On the other hand, non-beacons have no innate positioning capabilities, and therefore they can only estimate their positions through collaboration (i.e., trilateration). For more information on the roles and behavior of the devices, refer to Subsection 5.1.

Let us assume the existence of an imaginary town called Freeside, which is a well developed semi-rural town that holds a few governmental, residential, commercial, and educational infrastructures, as well as some amenities and parks. One of the town's perks is that it offers free Wi-Fi to all households of their population of 3,000 people, as well as several autonomous Wi-Fi hotspots throughout the town.

Sadly, Freeside has been hit by a 7.9 Richter scale earthquake. There has been extensive damage to its building, transport, and power structures, and although most civilian survivors have been evacuated, several more are still trapped under debris and collapsed buildings. In order to save them, a number of search and rescue teams have been deployed into the disaster zone.

Each team has been assigned a block or a set of blocks, depending on their size, and the teams scour the debris until they find survivors or not, and then they proceed to their next assignment. The rescue teams use geo-tagging to mark the buildings (or blocks) they are probing in a digital map, to show other teams whether there are survivors still trapped, the building is unstable and therefore it must be avoided, or it is clean and can be safely passed in subsequent searches.

Figure 4-12-a shows Freeside's town layout. Note that this section depends strongly on color coding; reading it in black and white is not advised. There are two Wi-Fi hotspots still working within the town, represented by yellow router icons. GPS, represented by a green satellite icon on the upper left side of the map, can still be used because the satellites were unaffected by the earthquake.

The relief-effort headquarters has been placed on the lower right section of the map, represented by a tent, and an antenna represented by a green icon has been placed next to headquarters to allow communication between command and rescue teams.

A total of five rescue teams have been deployed, represented by firemen icons in red, green, blue, purple, and orange. The teams are comprised of three to seven people, trained rescuers, firemen, military personnel, and volunteers. Each team carries a geo-tagging device used to mark buildings, which utilizes our model to perform the positioning estimations. In a real world scenario, far more devices would be deployed; we have included only five in our example for illustrative purposes only. Typically, at least a device mobile per team is required to keep communication and coordination among them and also with the command post.

Figure 4-12-b shows the connectivity between the devices while on the field, i.e., the

quality of the communication link between these devices. Connectivity is important because it is directly related to the response time and accuracy of the estimations; a good signal quality would yield more accurate measurements of distance (for triangulation) and lower delay times (for GPS utilization), while an unstable signal could require additional communication attempts or increase the error of the measurements due to environmental effects.

(a) Layout (b) Connectivity

Figure 4-12. Example of a MANET formed by rescue teams in a disaster scenario

The solid lines between nodes represent a good signal quality, while a dotted line represents unstable signals. Note that the distances and connectivity between rescue teams and reference points in Figure 4-12-b is not on scale, and they are shown as such for illustration purposes only. Some of the devices appear to be within range of others, but there is no connection between them; this is purposely done in order to emulate urban conditions, in which buildings and other obstacles could render a signal useless. Table 4-7 presents a summary of the teams' connectivity, as well as their capacity to perform positioning given their current context.

Table 4-7: Viability of positioning for the deployed teams

	# of Neighboring Devices		Positioning Capabilities	
	Good Signal	Unstable Signal	GPS	Collaboration
Red Team	2	1	Yes	Yes*
Green Team	6	1	Yes	Yes
Blue Team	3	0	Yes*	Yes
Purple Team	1	1	No	No
Orange Team	2	2	No	Yes*

* Connectivity with at least one transceiver is unstable, which translates into lower accuracy for the positioning estimation.

Now, let us return to our example scenario. Suppose that it is time for a change of shift, and headquarters command radioed all teams to update their positions and status. Each team would then use their devices to attempt to estimate their current position in order to do so. For ease of understanding, we assume that all teams will attempt to perform positioning simultaneously, during the course of several attempts. This is for visualization purposes only, because real world scenario devices perform asynchronous positioning attempts based only on their user's needs.

Additionally, the positioning attempts take place in span of seconds, so movement from the teams will not have a great impact over other teams' positioning attempts during collaboration. Table 4-8 summarizes each of the attempts, showing the current status of the teams' positions, the actions to be taken during subsequent attempts, and the result of executing the action.

At the beginning of the first positioning attempt of the teams, none of them knows their position. Thus, they attempt to use their mobile devices to update their status, by sensing the environment and detecting the elements pictured in Figure 4-12-b and summarized in Table 4-7 (i.e., for Green Team, there is GPS available, and six good neighbors and one unstable neighbor for collaboration).

Table 4-8: Status of the team's positions during several attempts.

# Attempt		Red	Green	Blue	Purple	Orange
1st	**Status**	Unknown	Unknown	Unknown	Unknown	Unknown
	Action	GPS	GPS	GPS (faulty)	N/A	N/A
	Result	Acquired	Acquired	Reattempt	Reattempt	Reattempt
2nd	**Status**	Known	Known	Known (faulty)	Unknown	Unknown
	Action	N/A	N/A	Collaboration	N/A	Collab. (faulty)
	Result	N/A	N/A	Acquired	Reattempt	Reattempt
3rd	**Status**	Known	Known	Known	Unknown	Known (faulty)
	Action	N/A	N/A	N/A	N/A	Collab. (faulty)
	Result	N/A	N/A	N/A	Hold	Hold
4th	**Status**	Known	Known	Known	Unknown	Known (faulty)
	Action	N/A	N/A	N/A	N/A	N/A
	Result	N/A	N/A	N/A	Hold	Hold

During the first positioning attempt, all teams but Purple have the potential to perform positioning. Purple team has no GSP access, nor enough neighbors for collaborative trilateration. However, since none of the teams know their position at the time, collaborative positioning is immediately ruled out, and GPS is asserted as the only viable option given the teams' current context.

Therefore, the Red, Green, and Blue use GPS, while Orange and Purple must reattempt the sensing in order to try detecting potential changes in their context. The Red and Green teams receive their estimated position without issue, but the Blue team's positioning error is too great to be useful (due to the unstable link, which represents

negative environmental conditions), and it must perform a new estimation.

During the second attempt, the Red and Green teams already know their positions, and their context has not changed (i.e., they have not moved), so their current position is updated using the latest stored. The Blue team, on the other hand, must attempt a new positioning estimation. It senses its environment again and detects the faulty GPS connection again, as well as three neighbors viable for collaboration.

Since GPS has already failed once and there is a new option, collaboration is designed as the strategy to be used, and thus the Blue team uses one of Freeside's remaining routers, the HQ's antenna, and the Red Team's transceiver (which now has a known position) as reference points to perform trilateration, successfully estimating its position.

The Orange team also has access to three viable neighbors during this second attempt, two of Freeside routers and Green team's transceiver, although one of the routers has an unstable connection. Collaborative trilateration is the chosen strategy, and although the Orange manages to estimate its position, it has a high error due to the faulty connection. Thus, the model determines that the device must reattempt to estimate its position in following attempts.

During the third attempt, the Red, Green, and Blue teams already know their positions, so the new estimation is the same as the last known one. As for Purple team, its device senses its context again and it remains unchanged, with no positioning strategy available.

Thus, the model tells Purple to put its estimations on hold for a time, in an attempt to avoid spending energy. It will wait a few minutes before trying again, in an attempt to allow its context to change (i.e., new neighbors could enter its range, and other conditions such as heavy clouds or rain could stop).

Orange team's device knows its position, but with low accuracy, so it makes another attempt at sensing its environment, finding no change in its context. Thus, Orange updates its current estimation using the last known one, stressing that it has low accuracy, and puts future attempts on hold for a few minutes in an attempt to save energy while waiting for significant changes in its context.

Nothing occurs during the fourth attempt of this scenario, given that Red, Green, and Blue teams already know their positions with acceptable accuracy, and Orange and Purple teams' estimation requests have been put on hold. All of the attempts should have been performed within a time window of up to one minute, depending on the conditions of the scenario. GPS has an average response time of 5 to 20 seconds [91], while trilateration could take between 10 and 30 seconds [36]. In both cases, the response time is small enough that the movement of the teams does not hamper the accuracy of the

collaborative estimations.

In the following sub-sections we describe in detail the context-aware positioning model, as well as the specific actions conducted during each of the stages that comprise it. For ease of understanding, all of the interactions between elements of the model are shown in the sequence diagram shown in Figure 4-3.

Chapter 5: Experimental Design

The purpose of this Chapter is to provide technical information on the inner workings of the simulated scenario, as well as showing how our proposed model interacts with the nodes during the simulations. It is intended for fellow researchers who would want to replicate our experiment, or improve this proposal. If the focus of the reader is only on the model, this section can be safely skipped, although several aspects and configurations of the model and the simulations are described within.

The experiments presented in this work have been conducted using a series of outdoor simulated scenarios, each populated with three types of nodes: stationary nodes, pedestrians, and vehicles. Some of these nodes have positioning capabilities (e.g., a smartphone with an embedded GPS transceiver), and they interact with each other through a mobile ad hoc network (MANET) to sense neighboring devices, and to share contextual elements with them.

All scenarios were designed using the third version of the Network Simulator 3 (ns-3 [67]), a networking simulator built as a core system with a set of interchangeable libraries that can be linked or imported to user programs (simulations). It provides substantial support for simulation of networking scenarios and it is highly scalable, extensible, and modular [67]. The versatility of the ns-3 allows users to write simulation scenarios (based on already validated models) using only the features they want to represent, as well as adding their own libraries to the core of the simulator.

The nodes mobility was modeled using the BonnMotion simulator [5], since this tool includes well-known models that represent people mobility in the study scenarios, particularly for disaster areas [4]. The mobility models considered in this work have a widely analyzed and validated mathematical basis [25, 72, 89], and they are accepted as valid real-world simulated counterparts. The randomized decision tree was implemented using an existing library, obtained from the OpenCV API Reference Library [124].

The following subsections describe the parameters of the simulated environment and the conditions under which the simulations are performed. They also explain the implementation of the experimental test bed, as well as the setup and use of the randomized decision tree library.

5.1. Experiment Description

The main goal of the proposed positioning model is to provide most or all devices in an ad hoc network with access to positioning, either by themselves or through collaboration, even if they have no positioning peripherals or strategies available, or they are under challenging conditions. In such a way, the model intends that the nodes spend no more energy than they would use under normal circumstances.

As stated before, all of the research presented in this thesis has been performed, evaluated, and tested by means of simulated scenarios emulating real-world outdoor environment conditions; all modeled using the ns-3 network simulator. The mobility and scenario setting models used in our simulations are well accepted by the research community as valid counterparts to real-world scenarios, and have been extensively used in networking research [54, 77, 84, 132, 138, 150, 155]. As such, the experiments presented in this thesis work can be considered as statistically identical to any experiment performed in the real world, as long as both have similar environmental conditions.

Our simulated scenario uses the IEEE 802.11b standard for Wireless Local Area Networks, and considers that all the participant mobile nodes are or can become part of a MANET. The simulated network allows for loss of packages due to collision or propagation loss (common in outdoor environments), as well as the retransmission of these packets [13]. To all extents, nodes with self-positioning capabilities do not need to communicate to determine their own positions, but they can still help other nodes that are impeded to estimate their positions due to environmental conditions or lack of proper peripherals. The latter nodes are restricted to exchanging positioning information with other nodes in order to determine their positions; if they are unable to “talk” with other nodes, they cannot perform estimations.

Under real world conditions, the greater the distance between an emitter and receiver, the more likely transmission errors will occur, represented by package losses, communication link severance, and retransmission requests, among others. To reproduce these adverse conditions, the ns-3 allows simulation designers to choose propagation loss models, which provide a predefined behavior for communication between nodes, and allow for a degree of configuration within the model’s boundaries. If the desired communication behavior requires further changes, they can be integrated into the model, although it would be preferable to choose another model instead.

The propagation loss model chosen for our work is the *“maximal range”*, in which the propagation loss is determined based on the distance between transmitter and receiver (i.e., the communication threshold), emulating out-of-range package losses. Any transmission received within the designed communication threshold is assumed to be received at maximum transmission power, while transmissions “received” beyond the threshold are automatically given a signal strength of −1000 dBm (effectively zero), and are therefore dropped. Note that the ns-3 assigns randomly determined signal strengths to each package transmission, which are stored in the trace file; the maximal range propagation model simply does not take into account these values, and instead treats them as absolutes (i.e., a transmission is either received or dropped).

The simulations observe thresholds of 30, 50, and 80 meters; any transmission received from a distance greater than the thresholds is assumed as lost, and any transmission within range is immediately received. However, this is not the case in real-world network communications. Even if packages are transmitted within the transceivers

range, they do not always reach their intended destination, due to environmental or cursory conditions, such as bad weather or occlusion, which result in package loss or reception delays.

Thus, we must add another layer to our simulation to emulate the delays and the drops. To that end, we use the "*random propagation*" delay model, which adds a random time reception offset to each transmission, following boundaries set by the user. For our simulations, we have determined 0.1 s and 1.0 s as the delay boundaries. We also use this random delay value to determine if a package is dropped by setting an additional threshold. If the transmission delay is equal or greater than 0.7 s, there is a 50% probability that the receiver drops the transmission. In addition, in order to avoid flooding the network with positioning requests and probing attempts, we only allow communication between nodes at a distance of one hop; any broadcast received at 2 or more hops is immediately dropped by the receivers.

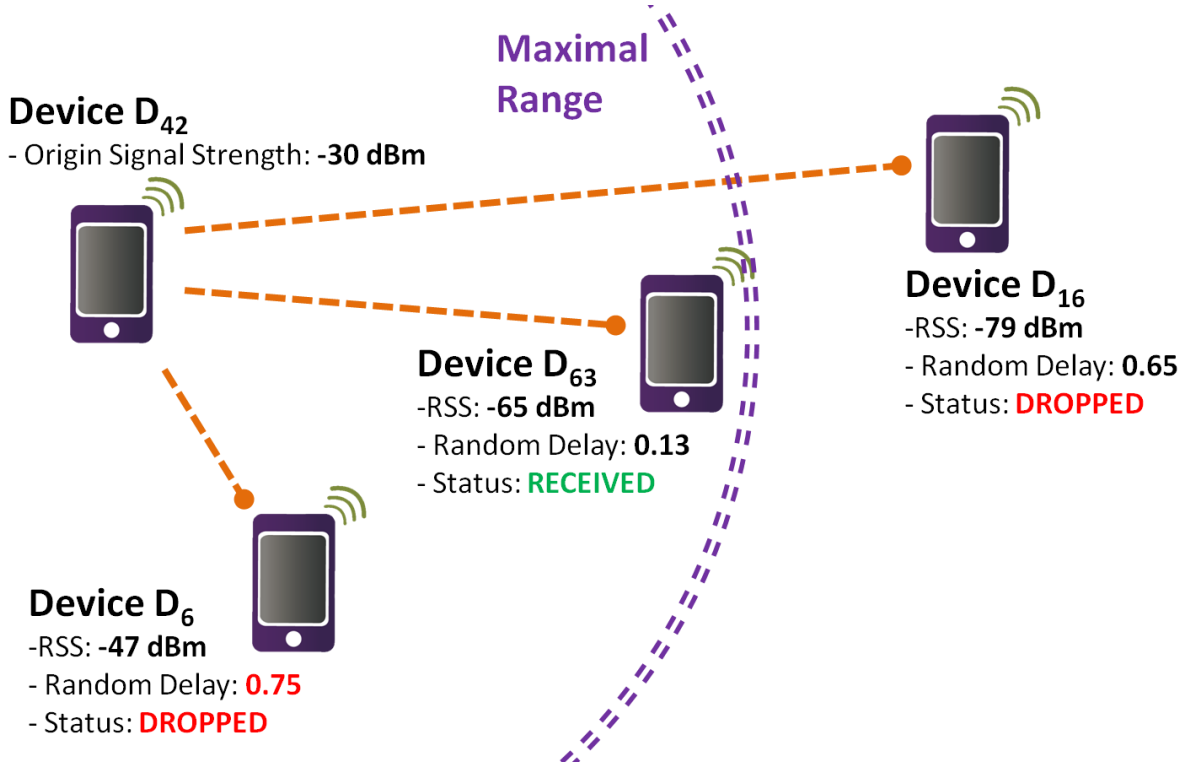


Figure 5-1. Behavior of the chosen propagation loss and delay models

5.1.1. Scenario Description

The simulations considered 500 × 500 m^2 areas, representing outdoor scenarios such as a park, a university campus, or a disaster zone, all of which are populated by three types of nodes: pedestrians, vehicles, and stationary (for more on node types, refer to Subsection 5.1.2). All scenarios can contain obstacles, such as walls, chasms, buildings, or trees; as well as roads for the vehicles to travel.

The scenarios are populated by all three types of nodes, in the following proportions: 65% pedestrians, 25% vehicles, and 10% stationary. This proportion was taken from a public transport road planning and design document from Western Australia [167]. The percentages represent a population of 70% pedestrians and 30% vehicles, with a small amount of each being completely stationary (e.g., people sitting on benches or having a picnic, or parked vehicles).

All of the nodes are placed within the scenario following different rules for placement (e.g., random placement, random from a list of viable positions, one by one from a list, etc.), and each node is assigned a random initial direction and speed values, based on their valid intervals. Moreover, each node is assigned a role, beacon or non-beacon (further discussed in Subsection 5.1.3), which determines their ability to perform positioning by themselves, or accessing a positioning strategy present in the environment.

At the beginning of a simulation, none of the nodes knows their position, and all have an energy level of 100% (specific maximum values vary depending on the simulated device brand). During the first few seconds, all nodes start sensing their context, attempting to estimate their position. Only beacon nodes are able to position themselves at first, but eventually non-beacons obtain positioning information from neighboring nodes and begin estimating their positions.

The first 30 seconds of a simulation are used for decay convergence, i.e., to allow the devices to mingle and update their decay indicators according to their perceived context. After that, we begin tracing four features: (1) the decay of the position of non-beacon nodes; (2) the number of non-beacons that know their positions, with a decay score of at most 60; (3) the number of non-beacon nodes that know their positions with any degree of decay (i.e., with decay scores over 0); and (4) the energy consumption of all individual nodes. The simulation last for 300 seconds after the decay convergence time, when the average scenario decay and amount of devices with known positions usually stabilize, at least until the energy levels begin to drop and the devices begin to shut down.

To validate the model, we have performed extensive tests under variable circumstances. The test battery included a total of 27 experiment templates, which were executed 15 times each for a total of 405 experiments. During each experiment, we observed the effect of the variation of the values of certain features, specifically changing the total number of nodes, assigning the beacon role with bias to certain types of nodes, and changing the maximum communication range of the devices.

We considered simulations with 100, 160, and 200 nodes; all using the proportions indicated earlier on this subsection. For the bias in the assignment of roles, we first assigned the beacon role randomly to any type of node; then only to pedestrian nodes; and finally only to vehicles. The communication ranges were assigned arbitrarily based on of actual ranges of different communication technologies.

In this work, we present the results of three different stages of the development of model, to show the improvement on its performance on the course of several benchmarks. For a summary and a discussion of the results, please refer to Chapter 6.

5.1.2. Types of Node

Regarding the nodes, pedestrians represent people on foot or using low speed transportation equipment (e.g., roller skates, hover boards), emulating the behavior of users engaged on their daily activities, such as walking around, shopping, or jogging, to name a few. This type of node can freely move throughout the entire scenario, their only restriction being that they cannot move through obstacles (e.g., walls or trees). Table 5-1 provides an overview of the behavior of each type of node.

Table 5-1. Node movement behavior comparison.

	NODE TYPE		
FEATURES	***Pedestrian***	***Vehicle***	***Stationary***
Speed	0.0 m/s, 1.0 – 4.0 m/s	0.0 m/s, 5.0 – 20.0 m/s	N/A
Steering Angle	0° – 360°	0°, 90°, and -90° (1)	0° - 360° (2)
Mobility Model	Random Walk	Random Waypoint	N/A
Movement Type	Free (3)	Bounded (3)	N/A

1 Vehicles are restricted only to keep going forward, and steering left or right.
2 True stationary nodes cannot rotate; however, pedestrian and vehicle nodes with an effective speed of zero can still steer while stationary.
3 Free movement implies the node can move in any direction within the scenario; bounded movement means that the node is restricted to certain sections only (i.e., roads).

The pedestrian's movement is based on the random walk mobility model [5], and they can change their direction (i.e., steer) in random angles and change their speeds within intervals of $[0, 2\pi]$ and $[1.0, 4.0]$ m/s, respectively. These changes occur after a time t has passed or a distance d has been covered by the node, depending on the how the mobility model is configured. Pedestrian nodes can also come to a complete stop (i.e., $0.0\ m/s$) for a time t, as people often do while pursuing their daily activities.

Vehicle nodes represent people riding bicycles, cars, motorbikes, or similar land transportation vehicles, emulating the behavior of users commuting through roads and roadways. Vehicle nodes move using the random waypoint mobility model [5], in which the nodes choose a random speed after a given waypoint has been reached; then, a new waypoint is randomly chosen, and the vehicle node resumes its movement.

However, our scenarios have a restriction that vehicles must only move through predefined routes (i.e., roads), and not freely. Thus, vehicle nodes can only steer in one of three ways; forward (no turn), right (90° turn), and left (-90° turn), and only if a waypoint

is available in the chosen direction. If a direction is chosen and no viable waypoint exists, it is discarded and another is chosen randomly. All roads in our simulations have been designed so that at least one steering option is available at all times. As for their speed, vehicles have a speed in an interval of $[5.0, 20.0]\ m/s$, and also could come to a complete stop (i.e., $0.0\ m/s$).

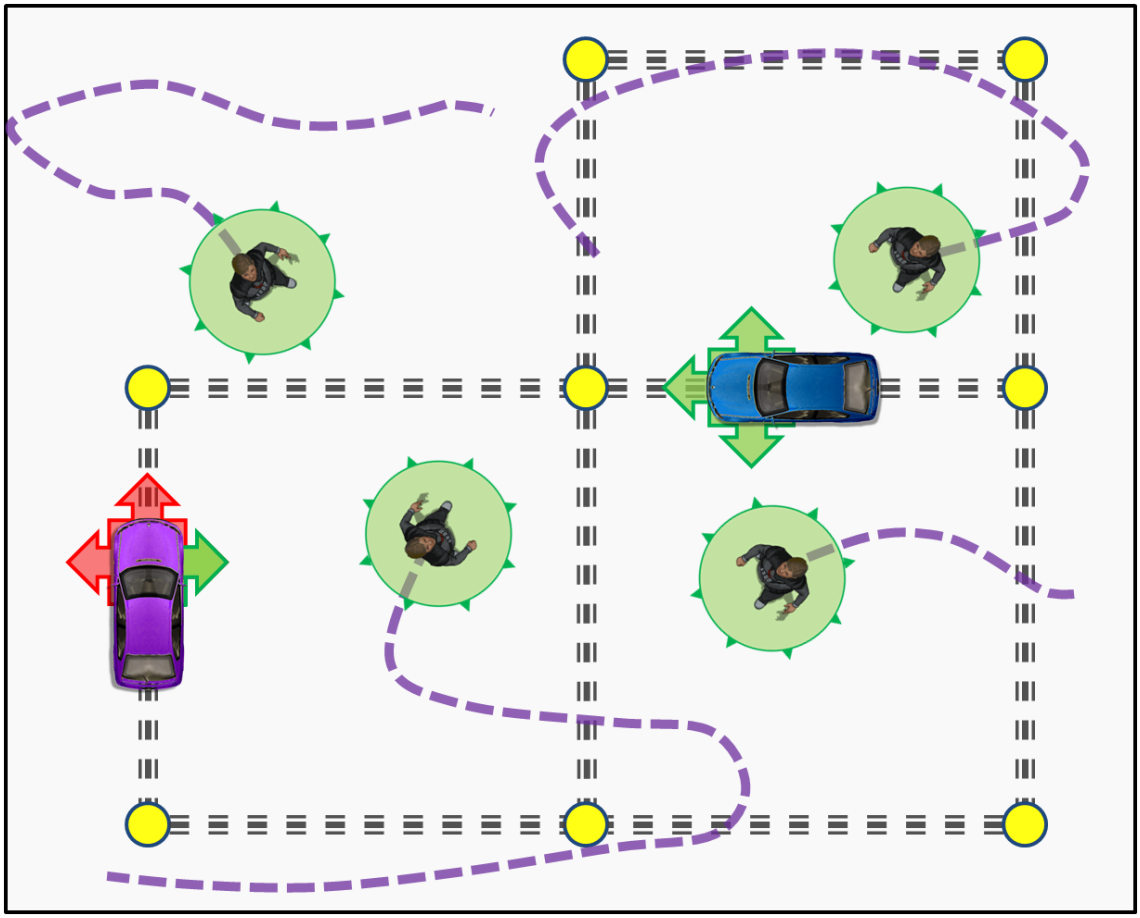


Figure 5-2. Pedestrian and vehicle nodes moving in a sample scenario. Pedestrian by Devin Night [40], cars from the CSUAC database [38]

Finally, stationary nodes represent base stations and devices that are immobile such as Wi-Fi Hot Spots, routers, or antennas. These nodes use the constant position mobility model [5], in which they have an effective speed equal to zero, although their direction can still change in the same interval allowed to pedestrians (this was hard-coded into the mobility model, and it could not be changed). In addition, stationary nodes are assumed to have access to recharging stations or other types of durable power sources, such as the electric network or power generators. This means that in our simulations, stationary devices have infinite energy, and therefore unlimited battery life.

Pedestrian and vehicle nodes that come to a full stop are considered as stationary, while their effective speed is $0.0\ m/s$. This is done to emulate users standing around or doing micro-mobility, such as parked vehicles, people sitting around resting, or working in

newsstands or hotdog carts. However, unlike “true” stationary nodes, which never move from their spot throughout the complete simulation duration, stationary pedestrians and vehicles do not have access to durable power sources, and thus retain their limited battery life.

5.1.3. Node Roles

In addition to their type, each node is assigned a role, which determines its positioning capabilities. Nodes can either be *beacons* (also known as anchors), or *non-beacons*. Beacon nodes have access to at least one positioning strategy (e.g., GPS), and therefore they are able to perform self-positioning when required by the users. Non-beacon nodes, on the other hand, have no positioning capabilities whatsoever, and they must rely on beacon and non-beacon nodes with known positions to determine their own position.

In order for non-beacons to determine their positions, a scenario must have at least 10% beacons present; otherwise, there might not be enough dissemination of useable positioning information, which would eventually lead non-beacons to stop attempting to position themselves, due to not detecting contextual changes over periods of time (refer to Section 4 for more on the behavior of the model). The threshold of 10% was chosen after analyzing our exploratory simulation results; thresholds with lower values led to average results where non-beacon positioning rates were close to 0% (for more on the subject, please refer to Section 6). For our experiments, we have considered three different configurations of beacons, only one of which can be used in a given simulation: 10%, 25%, and 50% of (beacon) nodes, respectively.

We assume that all participating devices are using the proposed context-aware positioning model. This means all nodes can potentially sense their environment and collaborate with their neighbors. The communication and navigation capabilities, represented in the simulation by the devices feature vector, are assigned randomly to all nodes. Thus, node A could have a feature vector showing that it possesses all communication protocols, three out of five communication technologies, no positioning capabilities, one navigation peripheral, *etc*., as shown in the following vector:

$$\Big[[1\ 1][1\ 0\ 1\ 1\ 0][0\ 0\ 0\ \ldots][[1\ 0\ 0]\ \ldots]\ \ldots\Big].$$

Meanwhile, node B could have only UDP, two communication technologies, GPS, two navigation peripherals, *etc.*, shown in the next vector:

$$\Big[[0\ 1][0\ 0\ 0\ 1\ 1][1\ 0\ 0\ \ldots][[0\ 1\ 1]\ \ldots]\ \ldots\Big].$$

This allows for a wide range of device configurations, and provides device heterogeneity to the simulations.

Since the scenario is highly volatile (i.e., most nodes are moving), we use the decay

indicator presented in Section 4 to validate the usefulness of the estimations, as time passes and the nodes move. Basically, the decay indicator begins with a value of 100 when a position is obtained, and then decreases based on the speed of a node over the course of a simulation. Once the decay reaches a threshold of 60, the model will begin requesting new positioning estimations; however, if the decay drops below a threshold of 20, the node's position is immediately set to unknown.

5.2. Implementation of the Simulated Scenario

The steps followed in order to build an experiment based on the ns-3 script mockup are detailed below. First we explain how we can represent the model in the simulation scenario using an application class; then we describe the behavior of the nodes and the mobility models that can be used; and finally we show how to set up channels for communication between nodes.

All of the algorithms shown in this section are presented in pseudo-code to facilitate understanding their functions. These scripts are useful to help in reproducing the experiments presented in this section, and also to allow researchers to add further improvements to the CAMPOS model.

5.2.1. ns-3 Application Setup

The ns-3 allows to easily setup and configure the behavior of any simulated scenario, by following a series of seemingly simple yet complicated steps. This subsection shows a detailed description of these steps, while configuring one of the most basic scenarios used in the early development of the model.

One of the first and most important steps in the implementation of the simulations is setting up the communication application, which will dictate how the nodes can behave during the simulations. This application also allows setting certain specific behaviors that trigger under certain circumstances, such as performing an action after moving a given distance, what to do after receiving a given message, and so on.

For our experiments, we required an application that simulated the exchange of mobile nodes' positioning information through a MANET. All nodes must be able to transmit and receive messages (in our case, positioning information and network stats), and they also have a decay score and a threshold indicator to help determine when a new positioning estimation must be performed. As we mentioned in previous Chapters, positioning is performed by the device, at the request of the context-aware positioning model. Thus, the communication application should only deal with sensing network elements, and requesting positioning information from other nodes; the actual positioning request is handled by the model.

Algorithm 5-1 shows an example of a common ns-3 application class. When this

application is attached to a node, the latter inherits all the functionality provided by the app, and thus it is able to communicate with other nodes to update its own decay decrement during each simulation turn, and to determine when a new estimation is required. The requests and positioning information are shared in the form of packets, and the decay information stored is a variable decrement, based on the time and the speed of the target nodes.

Algorithm 5-1. Application Class Example (C++ pseudo-code)

```
public class commApp

    public:
        commApp ();
        virtual ~commApp ();

        void Setup (socket, packetSize, packetAmount,
            packetInterval);
    private:
        virtual void StartSimulation ();
        virtual void StopSimulation ();

        void scheduleTransmission ();
        void sendPacket ();
        void updatePositionDecay();
        ***etc.***

        Ptr<Socket> m_socket;
        uint32_t    m_packetSize;
        uint32_t    m_packetAmount;
        Time        m_packetInterval;
        EventId     m_sendEvent;
        bool        m_running;
        uint32_t    m_packetsSent;
        uint32_t    m_decayDecrement;
        uint32_t    m_decayThreshold;
```

The Setup procedure designates which socket will be used to communicate (more than one could be used if several communication channels are available), the size of the packets, the maximum amount of packets that a node can send, and the transmission rate (i.e., interval) at which the packets are sent. The socket class acts as a handle (abstract reference) so that the network interface can determine the endpoints of a connection, e.g., "send this data (from origin) to the TCP 192.68.1.12:80 socket". One can view sockets as “the mouths and ears” of the nodes, to exemplify their use.

The packet size is static, because all of the nodes’ feature vectors have the same structure. The maximum amount of packages that can be sent should be set at high numbers (from 10,000 to 1,000,000, depending on the duration of the simulation) to ensure that the nodes do not “run out of packets”, and thus can continue communicating without issue. This variable can be used to limit the amount of messages that a node can send, but a new application must be created for each maximum packet amount, and then

attached to the relevant nodes.

The `StartSimulation` procedure tells the nodes that they can begin transmitting (when required), and listening to transmissions (all the time). `StopSimulation` does the opposite, shutting down all of the node's communication functionality. Once a node begins transmitting, the `scheduleTransmission` procedure takes over, and the application schedules the next execution of the `sendPacket` procedure at a time `t` in the future, and the node repeats the scheduling process as required.

The `updatePositionDecay` function triggers when the node moves, calculating the decay decrement based on the distance the node has moved, and the time duration of the movement for the given turn, so that the model can calculate the decay score of the node. Each decrement is stored with a timestamp and the node Id, so that we can always calculate the accumulated decrement of a certain time interval for a given node.

The `EventId` class serves as an identifier for the `Simulator::Schedule()` events. Each `m_sendEvent` variable is tied to a unique event, and it can be used to cancel or remove these events after they have been scheduled, by using `Simulator::Cancel()` or `Simulator::Remove()`. The `m_running` variable is simply an indicator of the state of the application; i.e., it is either currently active, or not.

The `commApp` class could also be used to setup triggers when sending or receiving messages, such as forcing nodes to drop certain communications, and ignore transmission from certain sources, as well as determining the conditions upon which it will begin or stop making communication requests. Other new procedures, functions, and variables can be added, and the existent could be modified or overridden, depending on what the user wants to achieve with his or her simulation. In addition, any number of applications can be attached to a node, as required, each representing a specific behavior.

5.2.2. Position Allocation

The ns-3 simplifies the overall configuration process of a simulation by providing embedded helper classes, which work like black-box interfaces. By using these helpers, the user can conveniently setup most of the scenario and node configurations with ease. Examples of helper classes include the `NodeContainer`, `MobilityHelper`, `WiFiHelper`, and `InternetStackHelper`, among others.

Algorithm 5-2 shows the `NodeContainer` helper class, which allows to create and store groups of nodes, which can then be batch-configured using the helper class instead of the individual node objects. Any permitted node configuration can be attached to the node container, and all the nodes within will receive that configuration; in addition, there is no limit to the number of node containers that can be created in a given simulation. The algorithm illustrates the creation of the three types of nodes: 65 pedestrians, 25 vehicles,

and 10 stationary, each stored in its own node container. Later in this Chapter, we show how to attach behavior to these helpers, providing them with different capabilities in terms of communication and movement.

Algorithm 5-2. Node Creation (C++ pseudo-code)

```
NodeContainer walkerNodes;
walkerNodes.Create (65);

NodeContainer vehicleNodes;
vehicleNodes.Create (25);

NodeContainer immobileNodes;
immobileNodes.Create (10);
```

Once the nodes are created and assigned to a container, they must be assigned a position within the scenario, which can be done using the `MobilityHelper` class. Algorithm 5-3 shows examples of three position allocators, `Grid`, `List`, and `RandomRectangle`.

Algorithm 5-3. Position allocators (C++ pseudo-code)

```
//Grid Position Allocator
mobility.SetPositionAllocator("ns3::GridPositionAllocator",
    "MinX", DoubleValue (0.0),
    "MinY", DoubleValue (0.0),
    "DeltaX", DoubleValue (distance),
    "DeltaY", DoubleValue (distance),
    "GridWidth", UintegerValue (10),
    "LayoutType",StringValue("RowFirst"));

//Random Rectangle Position Allocator
mobility.SetPositionAllocator("ns3::RandomRectangleAllocator",
    "X", StringValue("ns3::UniformRandomVariable[-250,250]"),
    "Y", StringValue("ns3::UniformRandomVariable[-250,250]"));

//List Position Allocator
Ptr<ListAllocator> positionAlloc = CreateObject<ListAllocator>();
    positionAlloc->Add (Vector (-210.0, 109.0, 0.0));
    positionAlloc->Add (Vector (-183.0, -202.0, 0.0));
    positionAlloc->Add (Vector (-211.0, 233.0, 0.0));
    positionAlloc->Add (Vector (-64.0, -112.0, 0.0));
    positionAlloc->Add (Vector (-45.0, 218.0, 0.0));
    ...
```

The `Grid` allocator, exemplified on Figure 5-3, arranges the nodes in a grid. The first node is placed at scenario coordinates `(MinX,MinY)`, and the rest of the nodes are allocated based on the chosen `LayoutType`; either `RowFirst`, or `ColumnFirst`. The `RowFirst` layout type places the next node at the same vertical coordinate (`MinY`) of the preceding one, separated by `DeltaX` units, until a total of `GridWidth` nodes are added

to the row. At that time, the next node is placed at the same horizontal coordinate of the previous row's first node (`MinX`), separated vertically by `DeltaY` units. The `ColumnFirst` layout acts similarly, but instead of placing the next node horizontally, it is placed vertically, separated `DeltaY` units from its preceding node, until `GridWidth` nodes have been placed. Then, a new column is created `DeltaX` units from the preceding column's first node, and so on. On either layout, the process continues until all nodes have been allocated.

The `RandomRectangle` position allocator creates a bounded area within the scenario, in which the nodes are allocated at random (hence the name). For each node, a random coordinate `(X,Y)` is chosen, where $X \in [MinX, MaxX]$ and $Y \in [MinY, MaxY]$. The process continues until all nodes have been added.

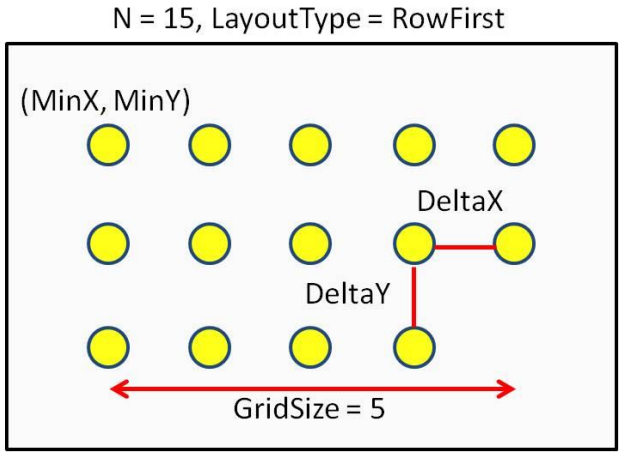


Figure 5-3. Grid position allocator description.

The `List` allocator allows the user to manually provide a list of predefined deterministic positions, which are assigned sequentially to each node by the simulator, in order of creation. If there are no positions left in the list, but there are still nodes left, the list resets and the remaining nodes are allocated in order from the beginning. If there are no more nodes left but there are unused positions in the list, the allocation process finishes normally.

It is important to choose position allocators carefully, as the results of a simulation are affected by them. The results from two different simulations using a `Grid` or `List` allocator could be very similar, while two simulations using the `RandomRectangle` allocator could yield very different results. Moreover, the `List` allocator, and in a lower degree the `Grid`, allow users to repeat experiments under identical position and mobility circumstances, which is useful to re-create scenarios while testing different values for the observed variables. In any case, once a position allocator has been chosen, it must be assigned to the nodes. To this end, simply create a `MobilityHelper` and assign the chosen position allocator to it, as depicted on Algorithm 5-4.

During our experiments, we have made extensive use of the `RandomRectangle` and `List` position allocators.

Algorithm 5-4. Using position allocators (C++ pseudo-code)

```
MobilityHelper mobility;
//Insert code of the selected position allocator (e.g., List)
mobility.SetPositionAllocator (positionAlloc);
```

5.2.3. Node Mobility

The next step in configuration of the scenario is determining how the nodes will move through the scenario. The ns-3 provides a comprehensive list of built-in mobility models, including random walk, random waypoint, constant acceleration, Gauss-Markov, etc. Depending on what the user wants to achieve, one or more of these models could be used in a simulation, each assigned to a specific `NodeContainer`.

Deciding which mobility model must be used by each group of nodes is crucial, because it greatly impacts the nodes ability to communicate with each other. In collaborative scenarios, the nodes' mobility determines the degree of dissemination of information throughout the scenario, since nodes with high mobility could spread messages between clusters of nodes that would otherwise be out of range from each other.

The following subsections detail some of the built-In mobility models offered by the ns-3, as well as importing mobility models from external tools, specifically from BonnMotion [5], a versatile mobility generation tool for different networking simulation environments.

5.2.3.1. ns-3 Built-in Mobility Models

Algorithm 5-5 shows examples of the three built-in models we have used in our simulations; `ConstantPosition`, `Waypoint`, and `RandomWalk2d`.

The `ConstantPosition` model is self-explanatory. Nodes using this type of mobility cannot move or change direction during the course of a simulation.

In the `Waypoint` mobility model, each node stops moving for a user-defined duration (`Pause`), after which it picks a new waypoint (via the `PositionAllocator`), and a new random speed. The node then moves at a constant speed, pausing again after the waypoint is reached, and starting the process all over. This mobility model enforces no bounding box by itself; it is the `PositionAllocator` assigned to the node that bounds the movement. If the user provides no pointer to a `PositionAllocator` to be used to pick waypoints, the simulation program will assert.

Algorithm 5-5. Mobility models (C++ Pseudo-code)

```
//Constant Position Mobility Model
mobility.SetMobilityModel("ns3::ConstantPosition");

//Random Walk Mobility Model (based on time)
mobility.SetMobilityModel("ns3::RandomWalk2d",
    "Bounds", RectangleValue(Rectangle(-250, 250, -250, 250)),
    "Speed", StringValue("ns3::UniformRandomVariable[6, 20]"),
    "Time", TimeValue(Seconds(1)),
    "Mode", StringValue ("Time"));

//Random Walk Mobility Model (based on distance)
mobility.SetMobilityModel("ns3::RandomWalk2d",
    "Bounds", RectangleValue(Rectangle(-250, 250,-250, 250)),
    "Distance", DoubleValue(30),
    "Mode", StringValue ("Distance"));

//Random Waypoint Mobility Model
mobility.SetMobilityModel("ns3::RandomWaypoint",
    "Speed", RandomVariableValue (ConstantVariable (200)),
    "Pause", RandomVariableValue (ConstantVariable (5)),
    "PositionAllocator", PointerValue (anyPositionAllocator));
```

Under the `RandomWalk2d` model, each node chooses a random speed and direction within the user-provided boundaries, until a stop condition is reached. If one of the scenario boundaries is hit, the node "rebounds" with a reflexive angle and speed. The `RandomWalk2d` stops condition (which forces a change of speed and direction) can be based on either `Time` or `Distance`.

In the case of Time, the nodes move for time `t` (in seconds) and then randomly change direction and speed. For `Distance`, the nodes move `d` units, and then change direction and speed randomly. In addition, if we require the nodes to follow a predefined path, we can assign direction and speed manually, and modify these values through triggers at predefined times.

Once the position allocator and mobility models have been chosen, they must be assigned to a node, or `NodeContainer`. The process is depicted in Algorithm 5-6. Only one allocator and one mobility model can be assigned to a single node, but any number of combinations of position allocators and mobility models can be assigned to any node during a single simulation; in addition, the mobility model of any node can be changed during the course of a simulation through triggers.

Algorithm 5-6. Assigning mobility models (C++ pseudo-code)

```
MobilityHelper mobility;
//Insert code of chosen mobility model (e.g., RandomWalk)
mobility.Install (anyNodeContainer);
```

As a side note, cutting off the initial phase is an important feature, because it allows the nodes to disseminate through the scenario regardless of their initial allocation. It has been observed that with the Random Waypoint model, nodes have a higher probability of being near the center of the simulation area, while they are initially uniformly distributed over the simulation area [67].

5.2.3.2. BonnMotion

On a side note, a versatile tool that can be used to create mobility models or modify the behavior of existing ones is BonnMotion [5]. BonnMotion is a Java software which creates and analyzes mobility scenarios for the investigation of mobile ad hoc network characteristics.

It is a versatile tool that allows exporting user configured mobility models scenarios in the format of several network simulators, including the ns-3. It is currently being developed by the Communication Systems group at the University of Bonn, Germany; the Toilers group at the Colorado School of Mines, USA; and the Distributed Systems group at the University of Osnabrück, Germany.

For any chosen mobility model, a series of parameters can be used to configure the behavior of the nodes. Examples include the amount of nodes (set with `-n`), the scenario duration in seconds (`-d`), and the convergence time, also known as initial phase or the time given to the nodes to "mingle" (`-i`). Algorithm 5-7 shows the creation of a Random Waypoint scenario populated by 160 nodes, and with a duration of 600 seconds, and an initial phase of 300 seconds.

Once the mobility scenario has been created, it is stored in two files: the first, with the suffix *.params*, contains the complete set of parameters used for the simulation, while the second, with the suffix *.movements.gz*, contains the (gzipped) movement data. Any parameter can be modified once the scenario has been created, by modifying the *.params* file accordingly.

Algorithm 5-7. Setting up the waypoint mobility model (Bash)

```
//Creating a mobility scenario from scratch
$ ./ bm -f scenario1 RandomWaypoint -n 160 -d 600 -i 300

//Creating a mobility scenario from an existing one
$ ./ bm -f scenario2 -I scenario1.params RandomWaypoint -h 5.0
```

The second line of code of Algorithm 5-7 shows how an existing scenario can be used as a template to create other scenarios. This is achieved by using the existing values for the parameters in the previous scenario (thus, a template), while being able to modify others directly through the command line. In the given example, the `-h` parameter, which represents *maximal speed*, is modified to $5\ m/s$, while the other parameters will remain the same as those of *scenario1.params*.

In order to use a BonnMotion generated mobility scenario, in ns-3, it must be first converted to the NSFile format, using the line of code from Algorithm 5-8, which generates two files as output. The first is the *.ns_params* file, which must be modified to set up some variables that are required by the ns-3, including the height and width of the simulated area, the number of nodes and duration, and the duration of the simulation, among others. The second file is the *.ns_movements* file, which schedules the movements of the nodes.

Algorithm 5-8. Creating an ns-3 compatible BonnMotion mobility scenario (Bash)

```
$ ./ bm NSFile -f scenario1
```

Once the *.ns_params* file has been setup, the *.ns_movements* can be called as a parameter when running an ns-3 TCL script via `-traceFile`, as shown on Algorithm 5-9. The `traceFile` parameter must point to the folder where the *.ns_movement* file is stored.

Algorithm 5-9. Importing a ns-3 compatible scenario from BonnMotion into an TCL script (Bash), and assigning it to nodes (C++ pseudo-code)

```
//The traceFile parameter can be added at the end of the call
$ ./ waf --run "scratch/scenario1
         --traceFile = /path/to/scenario1.ns_movements"

//The traceFile variable must be linked to a MobilityHelper, and
//installed to a single node, a group of nodes, or a NodeContainer
MobilityHelper mobility = MobilityHelper(traceFile);
mobility.Install (anyNodeContainer);
```

5.2.4. Communication Behavior

Once the nodes and the application have been set up, the scenario communication configuration variables must be configured. Algorithm 5-10 shows an example of how the environmental variables for a network are set up in an ns-3 script. In this case, the variables and values shown refer to the configuration of a Wi-Fi channel, which can be easily set through the `WifiRemoteStationManager` (referenced as "`ns3::Wifi`" in the example), `WifiHelper`, and `YansWifiHelper` classes.

As stated earlier on this Chapter, the scenario observes the IEEE 802.11b wireless standard, with a Request To Send/Clear To Send (RTS/CTS) handshake threshold set for frames above 2200 bytes, disabling fragmentation for frames under that value. Retransmission of lost packets due to collision or propagation loss is allowed [13], which means the nodes will make sure they can establish communication and then use a redundant protocol to make sure they receive packets correctly. The transmitted information consists of a device's contextual information (i.e., their feature vector), as well

as its unique node identifier.

Algorithm 5-10. Communication variables configuration (C++ pseudo-code)

```
double distance = 50;  // meters
int packetSize = 1000; // bytes
int numPackets = 10000;
Time interPacketInterval = Seconds(1.0);

//These are the default values for the WifiRemoteStationManager
Config::SetDefault("ns3::Wifi::FragmentationThreshold", 2200);
Config::SetDefault("ns3::Wifi::RtsCtsThreshold", 2200);
Config::SetDefault("ns3::Wifi::NonUnicastMode","DsssRate1Mbps");

WifiHelper wifi;
wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
wifiPhy.Set("RxGain",DoubleValue(0));
wifiPhy.SetPcapDataLinkType(YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
```

The NonUnicastMode variable enables easy switching of mode for broadcasting/groupcasting packets, which are sent at a DSSS rate of 1 Mbps. DSSS, which stands for direct-distributed-multiplexor-distributed-demultiplexor sequence spread spectrum, is a modulation technique that allows to transmit signals using a bandwidth that is in excess of the bandwidth that is actually needed by the message. The main idea of DSSS is that by spreading the transmitted signal over a large bandwidth, the resulting wideband signal appears as noise, which allows greater resistance to intentional and unintentional interference with the transmitted signal [65].

The RxGain variable allows to assign an intentional signal gain at the time of transmission, which translates into higher RSSI values at the receivers' end. It can be used to emulate the use of signal booster emitters, which are not necessary in our proposed environment; or to emulate stronger signals from certain devices. Note that for simulation purposes this value affects the signal speed, not the range, of the transmitter.

Next, the signal propagation delay and signal loss models to be used must be configured. The ns-3 offers several propagation models for both signal delay and signal loss, which represent the power fluctuations of the signal under different real-world conditions, such as environmental occlusion, signal shadowing, and multipath effects. Examples of propagation models include the Range Propagation Loss Model, Constant Speed Propagation Delay Model, Building Propagation Loss Model, and Okumura-Hata Propagation Loss Model, and the assignation of these models is shown in Algorithm 5-11.

Algorithm 5-11. Examples of propagation delay and propagation loss models (C++ pseudo-code)

```
//Constant Speed Propagation Delay Model
```

```
wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagation");
//Random Propagation Delay Model
wifiChannel.SetPropagationDelay("ns3::RandomPropagationDelay");

//Friis Propagation Loss Model
wifiChannel.AddPropagationLoss("ns3::Friis");
//Maximal Range Propagation Loss Model
wifiChannel.AddPropagationLoss("ns3::Range",
    "MaxRange", 50);
//Okumura-Hata Propagation Loss Model
wifiChannel.AddPropagationLoss("ns3::OkumuraHata");
```

For the range propagation loss model, the propagation loss depends only on the distance (range) in meters between transmitter and receiver, with the `MaxRange` attribute determining path loss. Receivers at or within range of the transmitter's receive the transmission at transmit power level, while receivers beyond the threshold receive it at power -1000 dBm, effectively zero.

The random propagation delay model, as its name implies, simply calculates a random delay in time for a transmission between two nodes. The constant speed propagation delay model, the signal speed is constant, independent of the distance between transmitter and receiver, i.e., there is no delay.

The building propagation loss model provides means for simulating shadowing (indoor and outdoor), external wall penetration loss, and internal wall penetration loss, when there are buildings present in an environment. Signal loss based on distance is deferred to derived classes that implement the `GetLoss` method, otherwise this model offers no distance loss estimations.

Finally, the Okumura-Hata propagation loss model takes into consideration the propagation loss caused in open, urban, and suburban areas, accounting for buildings, allowing path loss for distances of over 1 Km and frequencies ranging from 150 MHz to 2.0 GHz [104].

Propagation loss models can be used in conjunction with a propagation delay model, if they are compatible. The Friis propagation model gives the power received by one antenna under idealized conditions, given another antenna some distance away transmitting a known amount of power. The model uses the following equation:

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi d)^2 L}, \quad (3)$$

Where P_r is the reception power; $P_t$is the transmission power; G_t is the transmission gain; G_r is the reception gain; λ is the wavelength d; is the distance between transmitter and receiver (m); and L is the system loss.

The λ value is commonly calculated as C/f, where $C = 299792458$ m/s (the speed

of light in a vacuum); and f is the frequency (Hz), which can be configured by the user via the Frequency attribute. However, the Friis model is valid only for propagation in free space within the so-called far field region, which can be considered as approximately the region for $d > 3\lambda$. The model should still return a value for $d > 3\lambda$, as doing so (rather than triggering a fatal error) is practical for many simulation scenarios. However, the values obtained in such conditions cannot be considered realistic [88].

We tested results on a few other propagation models, until finally settling on using a modified version of the range propagation loss model, with a communication threshold of 50 m, representing the maximum communication range of the participant nodes. This propagation model is used in conjunction with a modified random propagation delay model, which adds a 0.7-second listening threshold to each transmission, which is used to randomly drop packages.

The acceptable amount of packet loss depends on the type of data being sent. For instance, losses between 5% and 10% of the total packet in a VoIP or video stream affect the quality significantly [100], but communication can still be maintained. On the other hand, when transmitting a text document or web page, a single dropped packet could result in losing part of the file, which is why a reliable delivery protocol would be used for this purpose (to retransmit dropped packets). We chose a threshold of 0.7 to randomly drop packages to account for the inability to change the individual range of devices; thus, a harsher drop rate needed to be instantiated.

Once the propagation loss and delay models have been chosen, they can be assigned to a communication channel (either a new or existing one) by means of the `YansWifiChannelHelper` class, as shown on Algorithm 5-12. The helper class `NqosWiFiMacHelper` allows to determine the MAC layer of the nodes (non-QoS), which can create MACs of the following types: access point (AP), non-AP station in a BSS (basic service set, or a single AP with associated stations), and ad-hoc (IBSS or independent BSS, where there is no central control AP).For our experiments, we use an ad-hoc MAC layer, and simulate AP and non-AP stations by including stationary nodes at certain points in the scenario.

The `ConstantRateWifiManager` station manager is set to use constant rates for all data and RTS transmissions, so that all packets are sent at the same transmission rate. The `DataMode` and `ControlMode` determine the transmission rate of data and control signals, which is set to 1 Mbps.

Algorithm 5-12. Configuring the global communication channel (C++ pseudo-code)

```
YansWifiChannelHelper wifiChannel;

//Code for chosen propagation delay model (only one can be active)
```

```
//Code for chosen propagation loss model (only one can be active)

wifiPhy.SetChannel(wifiChannel.Create());
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
    "DataMode",StringValue ("DsssRate1Mbps"),
    "ControlMode",StringValue ("DsssRate1Mbps"));
wifiMac.SetType ("ns3::AdhocWifiMac");
```

Once the communication channel has been configured, it must be installed into the nodes, and then these nodes must be assigned to a network. In order to configure the nodes' communication and network information in block, a `NetDeviceContainer` helper can be used. These helpers store a given `NodeContainer` and its associated channel and MAC (`YansWifiPhyHelper` and `NqosWifiMachelper`, respectively). Algorithm 5-13 shows both the installation of the communication channel to different types of devices.

Algorithm 5-13. Installing the channel (C++ pseudo-code)

```
NetDeviceContainer NDcontainer1 = wifi.Install(wifiPhy, wifiMac,
                                  nodeContainer1);
NetDeviceContainer NDcontainer2 = wifi.Install (wifiPhy, wifiMac,
                                  nodeContainer2);
NetDeviceContainer NDcontainer3 = wifi.Install(wifiPhy, wifiMac,
                                  nodeContainer3);
```

Now we must assign the node to a network, so that it can communicate with other nodes. In order to do this, the nodes require IP/TCP/UDP functionality, which is provided by the `InternetStackHelper` class. The helper takes care of setting up the traffic control layer, the communication protocol, and the routing strategy to be used (by default, it uses IPv4 with OLSR). The helper must then be installed to a node or `NodeContainer`, as shown on Algorithm 5-14, which assigns default networking options to the nodes.

Then, the `Ipv4AddressHelper` is used to set up the IPv4 settings, which are then assigned to the node as well. As a result, the node "belongs" to the designed network; e.g., the nodes in `NDcontainer1` belong to the network `10.1.1.0`. Note that this helper is simply a local incremental counter, and it does not assign IPs dynamically, so that all assigned IPs will be correlative for each `NodeContainer`.

Algorithm 5-14. Assigning nodes to networks (C++ pseudo-code)

```
InternetStackHelper internet;
internet.Install (nodeContainer1);
internet.Install (nodeContainer2);
internet.Install (nodeContainer3);

//Assigning IP addresses
Ipv4AddressHelper ipv4;
```

```
ipv4.SetBase("10.1.1.0,"255.255.255.0);
Ipv4InterfaceContainer interface1 = ipv4.Assign (NDcontainer1);
ipv4.SetBase(10.1.2.0",255.255.255.0");
Ipv4InterfaceContainer interface2 = ipv4.Assign (NDcontainer2);
Ipv4InterfaceContainer interface3 = ipv4.Assign (NDcontainer3);
```

Although our proposed positioning model supports different communication channels, due to ns-3 scenario limitations, we have focused on using Wi-Fi as the channel of choice, and simulated different channels (e.g., Bluetooth) by using different networks. The nodes can either use the same network, or separate networks with common nodes working as links (thus emulating the use of different channels), with a one-hop reception restriction for broadcasted signals.

After following these steps, the communication network structure of the ns-3 simulated scenario is finished. The nodes have some sort of mobility according to their type, and communication capabilities to represent their heterogeneity.

However, if the simulation is run as-is, these nodes will not communicate because they lack the means to do so, and thus will blind and deaf to other nodes. In order for the nodes do so, they must be assigned communication roles, either *sources* (emitters) or *sinks* (receivers). Sources are nodes with transmission capabilities, while sinks are nodes listening to transmission on a particular port.

For the experiments proposed in this thesis work, all nodes must be both sources and sinks, since they (potentially) have to collaborate with each other. To this end, we must create communication sockets on the nodes.

A socket, pictured in Figure 5-4, is one endpoint (i.e., an IP address and a port number) of a two-way communication link between two applications running on different nodes on a network. Sockets are bound to specific port numbers, so that the TCP layer can identify which application a specific socket is communicating with. Thus, each node would require at least two pairs of sockets; one pair for discovery/handshake requests, and the other to send and receive information packets.

As an example, picture a pair of radio handsets (i.e., devices). Any message sent is relied through the microphone, and heard through the speaker (i.e., sockets), allowing users *Ana* and *Bob* (i.e., the applications A and B shown in Figure 5-4) to communicate with each other.

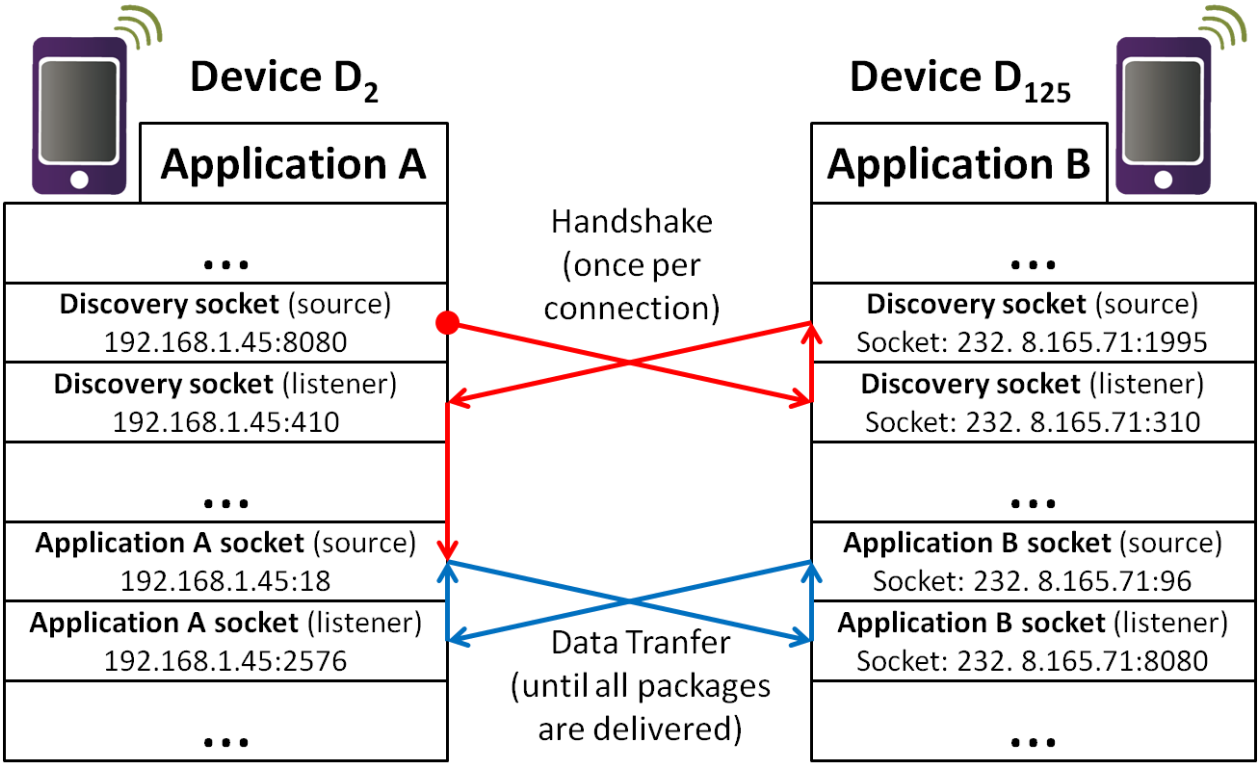


Figure 5-4. Socket description

Ana can send messages through her microphone on a certain bandwidth (i.e., a port), but they will only be received if Bob is listening to that particular bandwidth; otherwise, the message will be lost. If Bob is listening, he will hear the message through the speaker, and then send back an "acknowledge" through his microphone, beginning their conversation.

Algorithm 5-15 shows an example of the assignation of the roles of source and sink to the nodes. For the sinks, a socket must be created (`recvSink`) to listen for incoming messages, and an `InetSocketAddress` with a particular port must be assigned to it (`local`).

The `GetAny` option allows the node to listen to broadcasts from all sources, i.e., without network restriction. Then, a callback to the `ReceivePacket` must be set, to allow us to execute our code after a reception is successful. This process can be streamlined by using configuration methods, such as the `setupApps` method pictured in Algorithm 5-16, which shows pseudo-code examples of some methods implemented for our simulations. Any number of different applications can be added to different nodes, shaping the behavior of the nodes on different triggers, such as receiving a message, sending one, receiving messages from specific nodes, etc.

The assignation is similar for the sources. A socket is created, and an `InetSocketAddress` with a particular port is assigned to it, with the difference that it

must be assigned a particular destination IP, which in this case is the broadcast IP (255.255.255.255). With this, the node can broadcast messages to any listening node within range.

Algorithm 5-15. Source and sink assignation (C++ pseudo-code)

```
TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");

//Assigning Sinks to a group of nodes
Ptr<Socket> recvSink;
for(int iNode=0; iNode<65;  iNode++){
    recvSink = Socket::CreateSocket (aNodeContainer.Get(iNode),tid);
    local = InetSocketAddress (Ipv4Address::GetAny (), 80);
    recvSink->Bind (local);
    recvSink->SetRecvCallback(MakeCallback (&ReceivePacket));
}

//Assigning Source to a single node
Ptr<Socket> source;
source = Socket::CreateSocket (walkerNodes.Get (0), tid);
InetSocketAddress remote.Add(Ipv4Address("255.255.255.255"),80);
source->Connect (remote);
```

Algorithm 5-16. Examples of configuration and trigger methods (C++ pseudo-code)

```
//Application Setup
void MyApp::Setup (Ptr<Socket> socket, uint32_t packetSize,
uint32_t nPackets, Time pktInterval){
    m_socket = socket;
    m_packetSize = packetSize;
    m_nPackets = nPackets;
    m_pktInterval = pktInterval;
}

//Scheduling the beginning of the application operation
void MyApp::StartApplication (void){
    m_running = true;
    m_packetsSent = 0;
    m_socket->SetAllowBroadcast (true);
    SendPacket ();
}

//Scheduling the end of the application operation
void MyApp::StopApplication (void){
    m_running = false;
    if (m_sendEvent.IsRunning ()){
        Simulator::Cancel (m_sendEvent);   }
    if (m_socket){
        m_socket->Close ();  }
}
```

```
//Creating and sending a package
void MyApp::SendPacket (void){
    Ptr<Packet> packet = Create<Packet> (m_packetSize);
    m_socket->Send (packet);
    if (++m_packetsSent < m_nPackets){
        ScheduleTx ();    }
}

//Scheduling a transmission
void MyApp::ScheduleTx (void){
    if (m_running){
       m_sendEvent = Simulator::Schedule (m_pktInterval,
                     &MyApp::SendPacket,
                     this); }
}

//Receiving a transmission
void ReceivePacket (Ptr<Socket> socket){
  Address from;
  socket->RecvFrom(from);
  this->updateDecay()
}

//Streamlining the node's source and sink configuration
void setupApps(Node myNode, TypeId tid, InetSocketAddress remote){
    Ptr<Socket> source = Socket::CreateSocket (myNode,tid);
    remote = InetSocketAddress(Ipv4Address("255.255.255.255"), 80);
    source->Connect (remote);
    Ptr<MyApp> app = CreateObject<MyApp> ();
    app->Setup (source, 1040, 10000, Seconds (1.));
    myNode->AddApplication (app);
    app->SetStartTime (Seconds(double(rand () % 20 + 5)*0.001));
    app->SetStopTime (Seconds(600);
}
```

With this, the nodes have networking capabilities and they can dynamically form MANETs, but they will only use discover signals (pings) at engine-specified intervals, without sending any meaningful message. Basically, the nodes now see each other, but they are unable to talk because they do not know what to say.

In order to allow the nodes to exchange information, they must be assigned the communication application created in Algorithm 5-2, a task that is shown in Algorithm 5-17. In the pictured example, we use the `Setup` method from Algorithm 5-16 to assign the `source` node 10,000 packets with a size of 1040 bytes, which will be sent at a rate of one packet per millisecond.

The beginning and end of the transmissions are set via the methods `SetStartTime` and `SetStopTime`, at one and 20 seconds respectively, for the example.

Algorithm 5-17. Installing the communication application and

scheduling transmissions (C++ pseudo-code)

```
//Adding an app to a node
Ptr<MyApp> app = CreateObject<MyApp> ();
app->Setup(source,1040,10000, Seconds(.001));
walkerNodes.Get(0)->AddApplication (app);

//Scheduling the transmission
app->SetStartTime (Seconds (1.));
app->SetStopTime (Seconds (20.));
```

The next section details the classification process of the Randomizes Decision Trees, as well as their implementation.

5.3. Implementation of the Randomized Decision Trees

Random trees have been introduced by Leo Breiman and Adele Cutler [18], as a ways to perform classification and regression of data under certain parameters. The random trees ensemble, or forest, as it is more widely known, is a collection of tree predictors that takes a feature vector as input, then classifies it with every tree in the forest, and outputs the class that received the majority of votes. In case of a regression, the classifier response is the average of the responses over all the trees in the forest.

All the trees are trained with the same parameters, but on different training sets. These sets are generated from the original training set using the bootstrap procedure; for each training set, the same number of vectors as in the original set are randomly selected (N). The vectors can be chosen with replacement, which means that some vectors could occur more than once, while others could be absent.

At each node of each trained tree, not all the variables are used to find the best split, but rather a random subset of them. With each node in the tree, a new subset is generated, with the restriction that its size is fixed for all the nodes and all the trees, set to $\sqrt{number\_of\_variables}$ by default. None of the built trees are pruned.

In random trees there is no need for any accuracy estimation procedures, such as cross-validation or bootstrap, or a separate test set to get an estimate of the training error, due to the error being estimated internally during the training phase. When the training set for the current tree is drawn by sampling with replacement, some vectors are left out (OOB, or out-of-bag). The size of OOB data is about $N/3$.

The classification error is estimated by using this OOB-data as follows. For each vector, a prediction is calculated, which is OOB relative to the i^{th} tree, using the very i^{th} tree. After all the trees in a forest have been trained, for each vector that has ever been OOB, the winner class must be found (i.e., the class that got the majority of votes in the trees with vectors that fulfilled the condition) and compare it to the ground-truth response.

Then, the classification error estimate is computed as a ratio of the number of misclassified OOB vectors related to all the vectors in the original data. In case of regression, the OOB-error is computed as the squared error for OOB vectors difference, divided by the total number of vectors.

The library used for the experiments is the OpenCV 2.4.13.0 Random Trees, which can be found online at [124]. The following subsections briefly detail the process of setup, training, and execution of the library.

5.3.1. Setup and Training

The first step is to get the RandomTrees library from the repository and installing it. Then, the execution of the tree is as simple as linking the library to a script, setting up the variable values for forest, and calling the library's methods to train and execute the random forest classifier. In our case, the training data is a list of pre-assembled feature vectors with its resulting recommendation, and a single feature vectors for testing and for the actual classification.

A tree is built recursively, starting from the root node. The training data is used to split the root node by choosing the optimum decision rule (the best "primary" split) based on some criteria. In machine learning, "purity" criteria are used for classification, and the sum of squared errors is used for regression. If necessary, surrogate splits can also be found. All the data is divided using the primary and the surrogate splits between the left and the right child node, recursively splitting both left and right nodes [124].

At each node the recursive procedure may stop (that is, stop splitting the node further) if (1) the depth of the constructed tree branch reaches the specified maximum value; (2) the number of training samples in the node is less than the specified threshold (i.e., it is not statistically representative to split the node further); (3) all the samples in the node belong to the same class, or, in case of regression, the variation is too small; and (4) the best found split does not give any noticeable improvement compared to a random choice.

Once the tree is built, it may be pruned using a cross-validation procedure, if necessary. In that event, some of the tree branches that may lead to the model overfitting are cut off. Normally, this procedure is only applied to standalone decision trees. Usually tree ensembles build trees that are small enough and use their own protection schemes against overfitting.

Algorithm 5-18 shows the variable setup. The `max_depth` variable determines the depth of the tree. A low depth value will likely result in underfitting, and conversely a high value will likely lead to overfitting. The optimal depth value can be obtained using cross validation or other suitable methods, most likely trial and error.

Algorithm 5-18. Setting up the training the variables (Python pseudo-code)

```
rtree_params = dict(max_depth = 11,
                    min_sample_count = 5,
                    use_surrogates = False,
                    max_categories = 15,
                    calc_var_importance = False,
                    nactive_vars = 0,
                    max_num_of_trees_in_the_forest = 1000,
                    termcrit_type = (cv2.TERM_CRITERIA_MAX_ITER,1000,1))

classifier = cv2.RTrees()
classifier.train(train_data, //The list of [feature vector, class]
                     cv2.CV_ROW_SAMPLE, //tflag variable
                     label_data,
                     params=rtree_params);
```

The `min_sample_count` determines the minimum samples required at a leaf node for it to be split. A reasonable value is a small percentage of the total data (e.g., 1%). For our experiments, we tested different values, until we finally settled for 5%.

The `use_surrogates` variable determines whether surrogate splits will be built or not. A surrogate split is a mimic or a substitute for the primary splitter of a node, which allows to work when missing data is present (or rather, not present), computing variable importance correctly by using a close approximation (i.e., a clone) of a primary splitter that exists in the data. This is helpful when the classification process includes data that was not present during training; instead of crashing or exiting with error, the node determines a split based on previous information, and carries on.

In order to find a suboptimal split, `max_categories` allows the clustering of possible values of a categorical variable into $K \leq max\_categories$ clusters. If a discrete variable takes more than `max_categories` values during an attempted split, the estimation of the best subset estimation could take too long (the algorithm is exponential). Instead, the trees try to find sub-optimal splits by clustering all the samples into `max_categories` clusters that is some categories are merged together. The clustering is applied only in classification problems with more than two classes for categorical variables with $N > max\_categories$ possible values.

The `calc_var_importance determines whether` variable importance will be calculated or not during training. Variable importance determines the impact of the variables on the result.

The size of the randomly selected subset of features at each tree node and that are used to find the best splits, is set through `nactive_vars`. If set to 0, the size will be set to the square root of the total number of features; otherwise, it will be set to the users preference.

In an analog manner, the maximum number of trees in the forest is set through

max_num_of_trees_in_the_forest. Typically, the more trees you have, the better the accuracy. However, the improvement in accuracy generally diminishes and asymptotes past a certain number (cut factor), and also increases the prediction time linearly.

The final variable of the forest configuration process is termcrit_type, which determines the type of termination criteria. There are two termination types. The first, CV_TERMCRIT_ITER, terminates learning when the maximum number of trees in the forest is reached (max_num_of_trees_in_the_forest). The second, CV_TERMCRIT_EPS, terminates learning based on the forest_accuracy parameter (which indicates sufficient accuracy for OOB). In addition, both types can be used in tandem, entering CV_TERMCRIT_ITER|CV_TERMCRIT_EPS as the termination criteria. For our experiments, the termination criteria is CV_TERMCRIT_ITER, set to finish on 100 trees have been added to the forest.

Once the parameters for forest generation have been set, the training can begin. The classifier.train method receives four parameters, which are train_data, tflag, responses, and params. The method trains the statistical model using the input feature vectors (train_data) and the corresponding output values (responses), which are passed as matrices.

By default, the input feature vectors are stored as train_data rows, i.e., all the features of a training vector are stored continuously. However, some algorithms can handle the transposed representation, when all values of each particular feature (component/input variable) over the whole input set are stored continuously. If both layouts are supported, the method includes the tflag parameter that specifies the orientation as CV_ROW_SAMPLE to store the feature vector as rows, and CV_COL_SAMPLE to store it as columns.

The params variable stores miscellaneous parameters that might be required by underlying algorithms, or configuration parameters for the forest that could be required during execution.

5.3.2. Determining the Recommended Strategy through Classification

Once the training of a random forest has been completed, the process classifying is streamlined, as shown on Algorithm 5-19. The sample parameter represents a device's scenario feature vector, i.e., the target device's context. By executing the predict method over this sample, the cumulative result from all the trees in the forest (i.e., the class that receives the majority of votes) is returned as the chosen class for the given feature vector. And with that, a class has been determined for a given feature vector; i.e., a positioning strategy has been determined as the most suitable for the particular set of contextual features of the target device.

Algorithm 5-19. RT Classification (Python pseudo-code)

```
cv2.RTrees.predict(sample)
```

The following Chapter presents the results from our experiments, including our findings and a detailed discussion for two different scenarios.

Chapter 6: Analysis of Results

This Chapter is focused on showing the results that were obtained after analyzing the data from the simulations, followed by a discussion of the trends and indicators identified during this process. We have chosen to include the results from three non-consecutive stages in the development of the model, in order to show its evolution through different iterations of the research process.

The base experiment is thoroughly described in Section 5.1, and it consists on a series of simulations of mobile nodes attempting to perform positioning in outdoor environments under different conditions. The scenarios consist in 500 × 500 m^2 areas, which could contain obstacles (e.g., walls, buildings, and trees). Each scenario is populated by three types of nodes: pedestrians, vehicles and stationary nodes. The nodes are placed on the scenario following different rules (e.g., random placement, random from a list of viable positions, one by one from a list, etc.), and a random initial direction and speed is assigned to them.

Each node has a role, beacon or non-beacon, which determines their ability to use a positioning strategy present in the environment. When a simulation begins, nodes have an energy level of 100% and they do not know their position. During the first 30 seconds, all nodes start sensing their context, attempting to estimate their own position, allowing their decay indicator to converge (i.e., to change according to the context values).

During each simulation, we trace four features: (1) the decay of the position of non-beacon nodes; (2) the number of non-beacons that know their positions with a decay score of at most 60; (3) the number of non-beacon nodes that know their positions, with any degree of decay (i.e., with decay scores over 0); and (4) the energy consumption of all individual nodes. In the next sections we describe more in detail these simulations and present and discuss the obtained results.

The organization of this Chapter is as follows. First, we provide the results from the exploratory experiments, which were aimed at determining the working boundaries of the model. In Section 6.2, we present the results for three non-consecutive stages of the development of the model, and Section 6.3 provides the correspondent analysis and discussion for each of these sets of results. In Section 6.4 we provide answers to the research questions and discuss the validity of the hypotheses.

6.1. Overview of the Exploratory Experiments

This subsection shows the results from the exploratory phase of the experiments, in which we attempted to determine the effect of certain contextual variables over the performance of the model.

The first step was to assemble the simulation environment, a process thoroughly described in Section 5, and summarized at the introductory paragraphs of Section 6. The following step was determining the minimum acceptable value for the decay threshold of the last valid known position of a device, i.e., the minimum value at which a new positioning estimation is required.

The decay indicator (1), explained in Section 4.2, is the cornerstone of our model. It is a dynamic indicator of the validity of a positioning estimation, independent of its associated accuracy. When a position is successfully estimated, the decay indicator is set up at 100, and then steadily diminishes based on its associated formula, depending on time elapsed since the last positioning action and the distance travelled by the node. When the decay is low enough, the model attempts to estimate the position of the device again, by any means possible.

The following subsections provide the results of exploratory experiments we obtained by addressing specific contextual variables. First, we relate the series of events that let us to establish the decay and fitness indicators. Then, we describe the impact of changing the communication range of the participating devices. After that, we elaborate on the effect that the amount of nodes in a given scenario has on the results provided by the model. Finally, we discuss the effect of assigning the beacon role exclusively to either vehicles or pedestrians.

6.1.1. Decay Threshold

In order to determine the threshold for the decay indicator, we set up a single node with a random-walk mobility model, and let it run around, noting the different decreasing values for the decay. Figure 6-1 shows an early experiment, in which we run the same simulation 25 times, with a random decrease for the decay in the range of $[0.0, 10.0]$ for each second that passed.

Each time its decay decreased, the node had a hard-coded 50% probability of finding from one to five “neighbors” viable for collaboration (remember that this node was in fact alone in the scenario). This allowed the target node to make a bogus positioning estimation by averaging the decay values of its neighbors, which resulted in the disastrous values that can be observed in Figure 6-1. After analyzing the results, it was clear that a random decay decrement was not at all helpful, and that decay by itself was not a good indicator of whether the node was fit to perform positioning or not. The maximum decay value that could be reached this way was of 10, and the average was below 5. Thus, the notion of “fitness” was incepted.

Over the course of several additional experiments, we settled on the first formula for decay, which was dependent on distance traveled by the node and time elapsed since the last positioning. This approach to decay decreases the confidence on a positioning estimation according to the node’s own mobility, which made the scenario more dynamic

and allowed decay to stop decreasing when a node is stationary, increasing the usefulness of positioning estimations that would otherwise be labeled as deprecated.

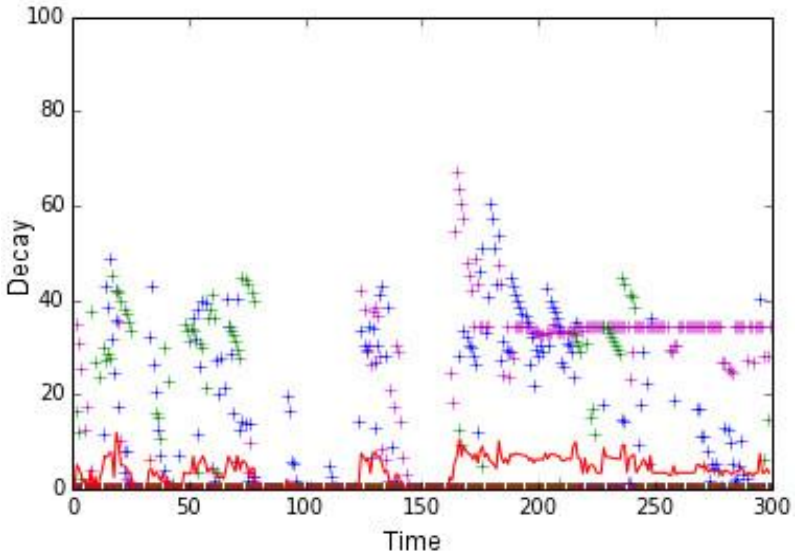


Figure 6-1. Decay measurements of a test node

As a side-effect, tinkering with the decay threshold helped us come up with the precursor for the *fitness* indicator, which allows determining whether a device is apt or not to perform positioning. If the fitness of a device is too low, it would not be taken into account for the positioning process, helping thus save energy by avoiding repetitive positioning attempts. In addition, during collaboration, the fitness indicator determines the order in which nodes are selected; those with greater fitness always go first. For self-positioning, by taking into account the accuracy requirement of the application, the device can opt not to perform positioning again if it obtains low fitness values several times. Fitness would gain more importance than decay regarding the model due to this.

After about 32 experiments, each consisting of 50 simulations, we settled on the decay formula shown in (2), and the fitness formula shown in (1) (page 53, Subsections 4.2.1.1.1 and 4.2.1.1.2), which provided a good positioning rate while maintaining the energy consumption levels at acceptable values.

These values are summarized in Table 6-1 for each type of node. We have differentiated between beacons and non-beacons, and the Table shows results of non-beacons only. This is due to beacons heavily influence the positioning rate and decay values, which could mean that the energy consumption may or may not have a little bias towards lower values. Stationary nodes do not count towards the global energy consumption, due to the assumption that these devices have access to a source of unlimited energy (as it is the case with access points, or devices plugged to the electric network).

We decided to go with *average* decay instead of *median* decay, due to the volatility

of the results from one simulation to another. Although the simulations tend to yield similar results when using the same scenario configuration (except for the random starting position and direction), for some sets of results the median was deceptively better than the mean, but in fact, it did not reflect the actual state of the scenario decay.

Table 6-1: Average observed values for a simulation (300 seconds).

	Type of Node		
	Pedestrian	**Vehicle**	**Stationary**
Average Decay	47	18	100
Positioning rate (per minute)	12	27	0
Energy consumption	~2031 mW	~4638 mW	N/A
% of nodes with Decay > 60	25%	8%	100%
% of nodes with Decay > 20	72%	45%	100%

In any case, the average decay values for pedestrian and vehicles are 47 and 18, respectively. Although these values are generally below our threshold of 60, one must consider that they are average not only for a single node, but for all nodes of a specific type in the simulation scenario. Note that the percentage of nodes with a decay greater than 20 includes the nodes with a decay greater than 60. For instance, the average decay of pedestrians is 47, because there are far more nodes having decay values between 20 and 60 than over 60 (47% and 25%, respectively, for a grand total of 72%).

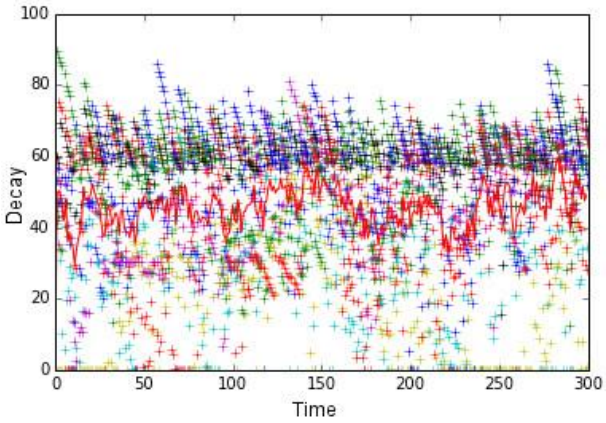
As for vehicle nodes, they consistently showed poor decay values on average for all simulations. It does not mean, however, that they did not know their positions; it was expected, as their decay diminished quickly due to their speed. Therefore, this situation is translated into more positioning requests than those of pedestrians, i.e., a higher positioning rate per minute, and therefore higher energy consumption.

The positioning rate values shown in Table 6-1 determine how many positioning estimations were requested on average by the corresponding type of node. Vehicles tend to request more estimations than pedestrians, which is understandable due to they usually move to high speed, which cause decay to deprecate quickly.

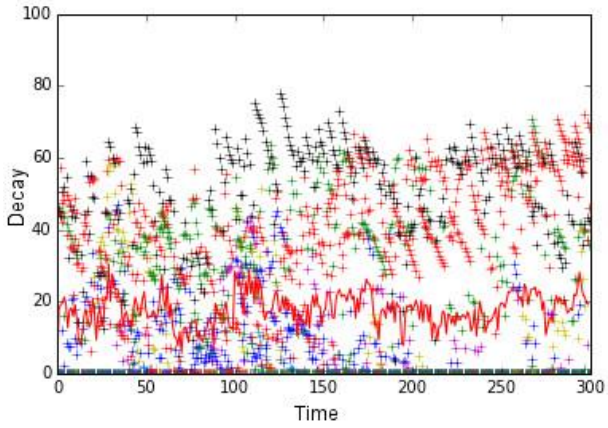
The average consumption of GPS is stated by Carrol et al. [28] as $166 \pm 0.04\%\ mW$. This value represents the consumption of an external antenna in a worst case scenario, and we have used that value as a reference for the consumption of the GPS internal antenna of a common smartphone. A rough conversion has been made from *Ah* (Ampere-hour, or the cost of providing 1 ampere continuously during one hour) to *mW* (milliwatts) to facilitate the comparison of energy consumption.

Figure 6-2 shows the decay results of three selected nodes, in a scenario populated following the rules for node type, proportion, role, and mobility, established in Section 5.

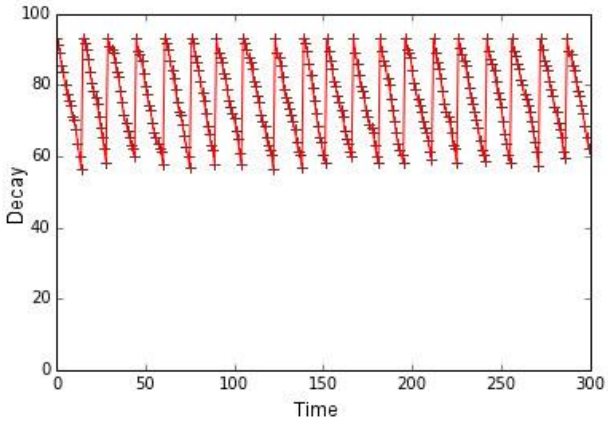
We found that non-beacon pedestrian nodes, such as the one shown in Figure 6-2-a, tend to have more stable decay values, in average. This is due to their limited speed, which allows them to remain within similar contextual conditions for longer periods of time, i.e., if they manage to find suitable neighbors, it is more likely they will remain together for longer due to their lower speed.



(a) Pedestrian non-beacon node (N_{47})



(b) Vehicle non- beacon node (N_{78})



(c) Pedestrian beacon node (N_{31})

Figure 6-2. Individual decay measurements of nodes of different type and role

The same cannot be said for vehicles, for the same reason we have explained earlier, i.e., due to their speed. All vehicle nodes, such as the ones shown in Figure 6-2-b, tend to have consistent lower decay values on all scenarios. As we stated earlier, it does not mean that they are unable to estimate their positions collaboratively, only that their decay (and therefore their fitness to participate in the collaboration process) decrements quickly due to their high speeds.

For comparison purposes, Figure 6-2-c shows an example of the average decay curve of a pedestrian beacon node. When the decay of the node decreases below the threshold of 60, the node simply attempts a new estimation, and updates its decay accordingly.

6.1.2. Communication Range

The communication range of a device determines the maximum distance at which such a device can communicate with other devices in a direct manner, i.e., without need for message routing. We have considered three different communication ranges for our experiments: 30, 50, and 80 meters. Due to limitations in the simulated scenario configuration, we had to assign the chosen range to all of the participant nodes, which means that all nodes have the same range for a given simulation.

We found that the length of the communication range has a direct influence over the amount of neighboring nodes that can be reached by the requesting nodes. As a rule of thumb, the greater the communication range of the devices, the more neighbors that can be reached. This ensures that the target nodes are able to estimate their positions more often than with lower ranges, even when the scenario is populated with low numbers of nodes.

Table 6-2 shows the average connectivity for the four considered amounts of nodes in a scenario: 100, 130, 160, and 200 nodes. These values have been obtained using a proportion of 25% beacons per scenario, i.e., 25 beacons for an amount of 100 nodes, 32 beacons for 130 nodes, and so on.

Table 6-2: Average node connectivity in a 25% beacon-node proportion scenario.

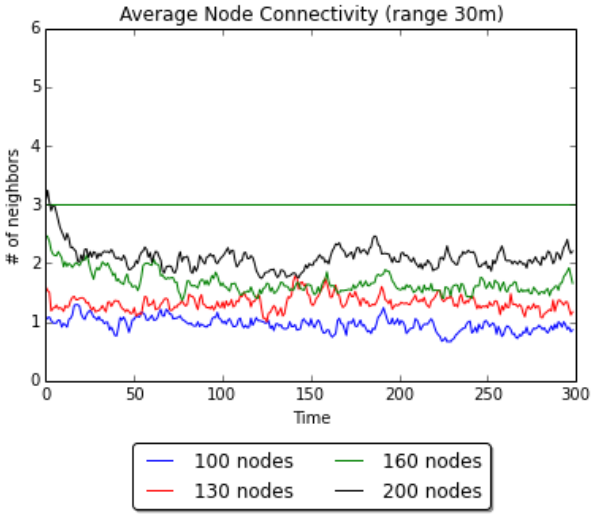
	Amount of Nodes			
Range (m)	**100**	**130**	**160**	**200**
30	1	1	1	2
50	2	3	4	5
80	5	7	8	10

For instance, for a range of 30 meters, the (rounded) average amount of reachable neighbors is of 1 for an amount of 100 nodes (on average). This means that for simulations populated by 100 nodes (following the proportions and rules stated in Section 5), any node has at least one neighbor at any given time, on average.

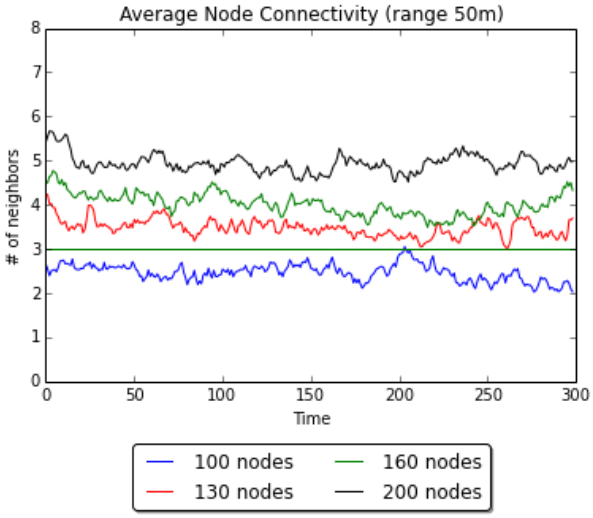
The trend is clear, and as the communication range increases, the connectivity of the nodes also increases. Figure 6-3 illustrates the values obtained after averaging results from a set of experiments. For all images, a threshold has been marked at the minimum amount of nodes required to perform triangulation, i.e., three (3).

Considering all the variables we tested, the communication range showed a greater effect over the node connectivity, effectively doubling the amount of nodes that could be reached by a requesting non-beacon for each range distance increment. In order to exemplify this point, in a scenario with 130 nodes, when the range is increased from 30

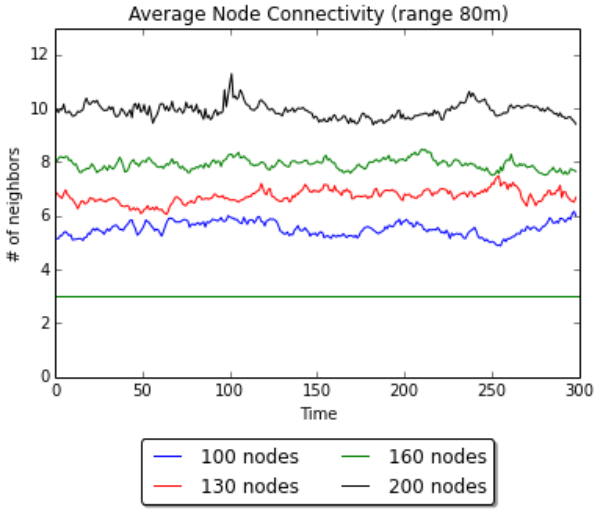
to 50 meters, the amount of reachable nodes doubles from 1 to 3 (rounded), and when increasing the range from 50 to 80 meters the connectivity doubles from 3 to 7 (rounded).



(a) Communication range of 30 m



(b) Communication range of 50 m



(c) Communication range of 80 m

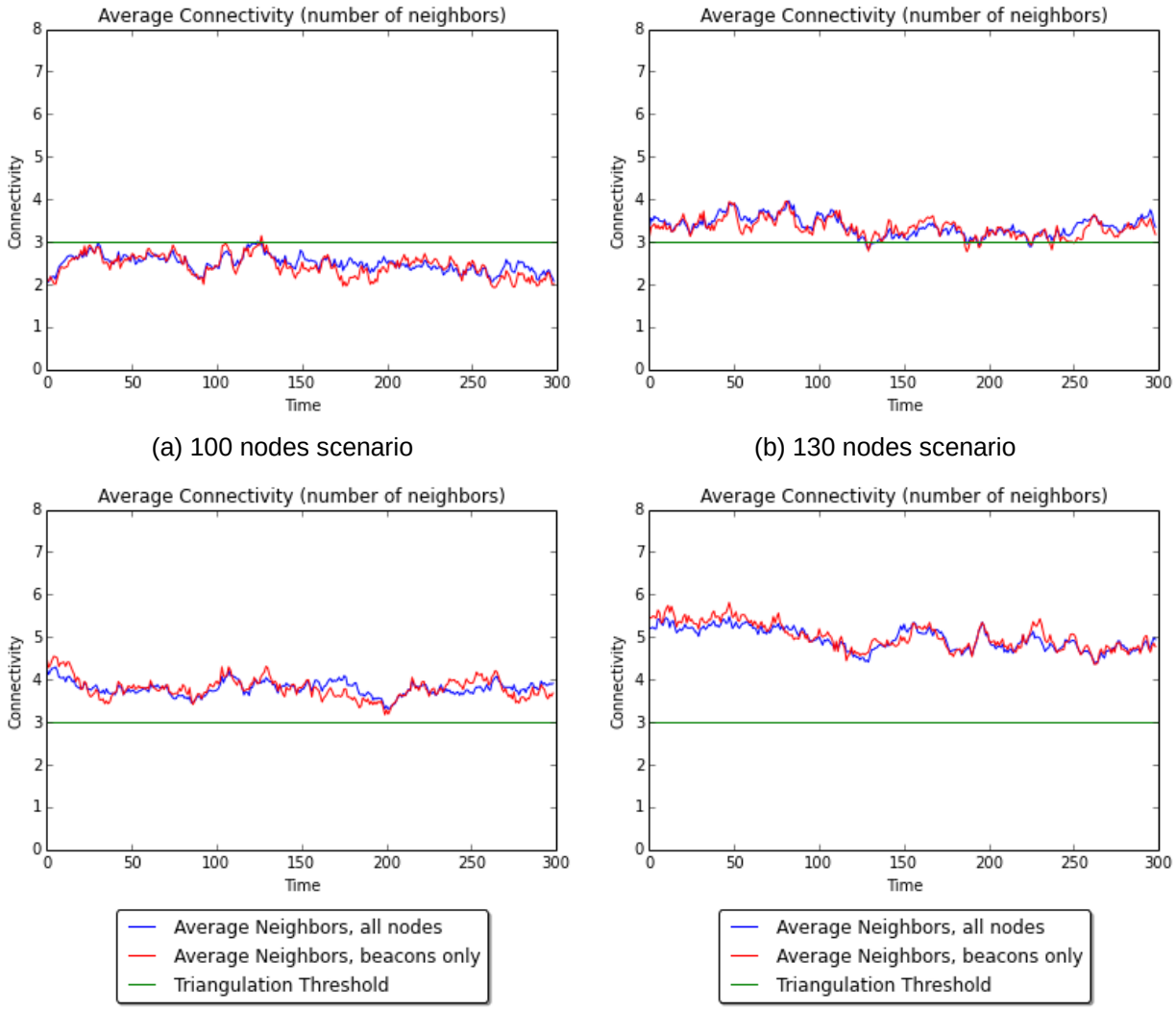
Figure 6-3. Average node connectivity over different communication ranges

6.1.3. Amount of Neighbors

The number of beacon nodes in a scenario is directly related to the probability of a requesting non-beacon finding neighbors that know their positions. When a scenario is populated by greater amounts of nodes, these petitioner nodes are more likely to find neighbors that know their positions (either beacons, or non-beacons with known positions), than in scenarios with lower population density.

Figure 6-4 shows the results of measuring the average connectivity of nodes in scenarios with a proportion of beacon nodes of 25% (i.e., 25 for the 100 node scenario,

32 for the 130 node scenario, etc.), a range of 50 meters, and the node proportion and mobility stated in Section 5.



(a) 100 nodes scenario

(b) 130 nodes scenario

(c) 160 nodes scenario

(d) 200 nodes scenario

Figure 6-4. Average node connectivity over different amounts of nodes

For each graph, two curves are shown. The blue line shows the actual average connectivity in the scenario, for all nodes (non-beacons and beacons). On the other hand, the red line shows the connectivity of beacon nodes only. This allows us verifying that the beacon nodes are actually reaching (and thus being reached by) neighboring nodes, disseminating their positioning information throughout the scenario as they move.

As expected the two curves overlap, which is a good indicator that the connectivity is stable for the entire scenario, in average. This is proved by the fact that the behavior of both curves is more similar as the number of nodes in the scenario increases, which makes sense given that the curves show average values for the nodes in the scenario.

Increasing the amount of nodes allows to provide better results in terms of

connectivity. The explanation of this behavior falls back on the amount of beacon nodes present in the scenarios. Since we follow beacon role assignation proportions, the greater the number of nodes, the greater the number of beacons, and therefore the more viable neighbors will be available for collaboration.

Finally, although the number of nodes does increase the connectivity of the scenario, this finding was not as crucial as that of the communication range. Moreover, as more nodes are present in the scenario, the energy consumption level of the entire network tends to grow (which is not a good thing), skyrocketing when less beacons are present and the communication range is too low.

6.1.4. Proportion of Beacons

This experiment was implemented to (1) observe how the amount of nodes with positioning capabilities in a scenario (i.e., beacons) affected the results of a simulation, and (2) whether the assignation of this role to specific types of nodes had any effect over the results.

For the first part of the experiment, we tested the scenario using the proposed communication ranges, proportions of beacons, and amount of nodes. The results, shown in Table 6-3, were conclusive. As expected, the proportion of beacons in a scenario directly increases the average decay level of the entire scenario, and it is affected by both the communication range and the amount of nodes in the scenario.

Table 6-3: Average decay based on amount of nodes and beacon proportion.

Beacon Proportion	Comm. Range (m)	Amount of Beacons			
		100	**130**	**160**	**200**
10%	**30**	13	15	16	18
	50	18	21	24	33
	80	34	36	37	39
25%	**30**	30	32	37	40
	50	37	39	44	53
	80	58	59	59	61
50%	**30**	43	45	47	48
	50	61	62	62	64
	80	67	67	68	71

It is important to note that the greater increment in the decay values is brought by the communication range, which varies ranging from 5 to 15 points. In contrast, the increase brought by the amount of beacons ranges from 1 to 9 points only. Interestingly, the total increment from the worst to best case scenarios was more favorable for the amount of beacons (+58) than for the communication range (+54), even though average increments were of +23 for the communication range and +7 for the amount of beacons.

In any case, these results served to help us determining the operating boundaries of

the model, in terms of device specifications. In addition, we found what we call a “standard scenario”, which is the scenario configuration that resembles the most to a real-world outdoor environment. This scenario observes a proportion of beacons of 25%, with a device communication range of 50 meters. We still tested all possible combinations, but we stuck to this particular scenario when deciding to make changes to the model.

For the second part of the experiment, we assigned the roles of beacon only to specific types of nodes. First, the role was assigned only to pedestrians, then vehicles, and finally stationary nodes. Table 6-4 shows the resulting average decay when assigning the role of beacon under different conditions.

Table 6-4: Average decay based on amount of nodes and beacon assignation.

	Amount of Nodes			
	100	**130**	**160**	**200**
Random Assignation	37	39	44	53
Pedestrians Only	32	33	40	47
Vehicles Only	12	15	16	21
Stationary Only [(1)]	0	0	1	4

[1] Stationary nodes are never taken into account when calculating the decay average, to avoid bias.

The first set of results is for random assignation, or the default assignation scheme explained in Section 5. The values shown in this table correspond to those of the standard scenario, with a 25% of beacons and a communication range of 50 meters. We will not discuss these results, as they have been analyzed previously.

The results for pedestrian nodes are very similar to those of the random assignation (a difference of only 5 points on average); therefore, we can safely assume that there is no benefit in assigning them the role exclusively. However, it is important to note that these results are still lower than those of random assignation, which was an indication that the heterogeneity of movement was in fact helping increase the decay values.

As for vehicle nodes, we expected that their high mobility would give them an advantage, allowing them to share their positioning information better than pedestrians, since they can travel faster throughout the scenario. We expected that this would translate into greater (better) decay levels; however, as shown in Table 6-4, we could not have been more wrong. Given that the decay formula depends on time and distance, the decay values of vehicles rapidly decreased, resulting in short bursts of high decay levels, followed by periods of lower decay. Even though the average decay levels of pedestrian nodes were well within acceptable ranges, vehicles lowered the scenario average faster than they could keep raise it. In addition, under this role assignation scheme, the energy consumption skyrocketed, due to the additional number of positioning requests performed by vehicles.

Finally, stationary beacon nodes proved to be the worst case scenario, with the

greatest average decay reaching only four points. This is due to the fact that the stationary beacons are few (10% of the total population) and they are not necessarily close by, which means that non-beacon nodes could have been unable to reach the minimum number of beacons required to perform trilateration. Given that the scenarios are quite large (500 × 500 m^2 sized) and the communication range is quite short (50 meters), a grid with a minimum of 5 stationary beacons per 100 × 100 m^2 section would have been required to ensure coverage for non-beacons. This is not only unlikely in outdoor scenarios, but it breaks our requirement of not needing specific instrumentation for CAMPOS to work.

Summarizing, these experiments allowed us determining a standard scenario, which we would later use to test our model, and also helped us realize that the assignation of roles was critical to achieve our goal of providing positioning for all devices in a scenario. With this, we have finished our presentation of the results from the exploratory experiments. Next section deals with the results from the actual experiments with the model.

6.2. CAMPOS Results

In this section we provide the results from two iterations of the CAMPOS development. The first set, shown in Subsection 6.2.1, includes the results from an early version of the model, which was built upon rule-based heuristics (i.e., a static decision tree). The second set, shown in Section 6.2.2, is from the latest version of the model (and regarding this thesis document, the last), in which we use the random forest classifier to assemble the recommendation vector of suitable positioning strategies.

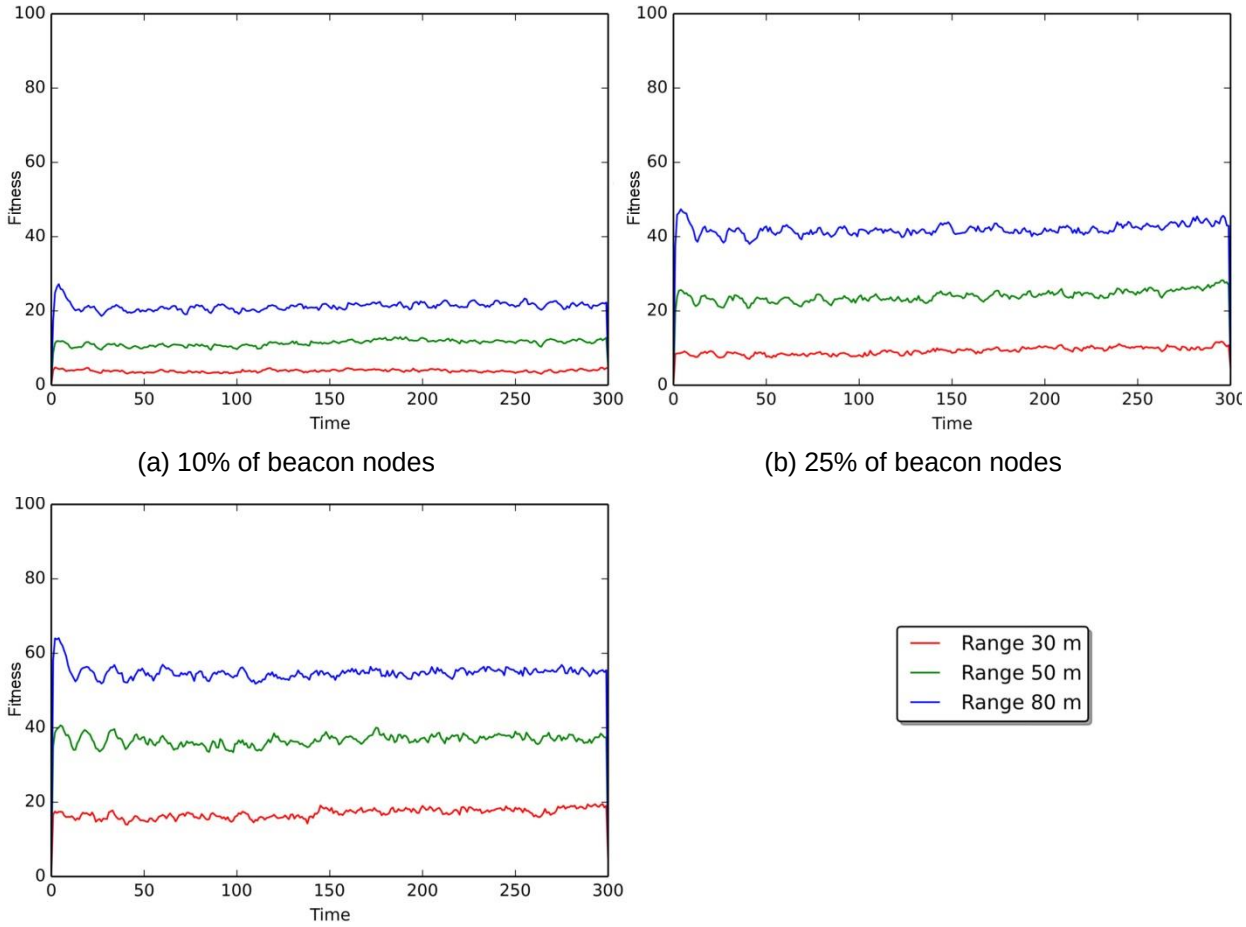
The results obtained during different stages of the development of the model have been published in the IEEE Systems, Man, and Cybernetics Conference (SMC) [112, 113], and the Sensors Journal [114].

6.2.1. Early Model Development: Rule-based Heuristics

As described in Section 5, the objective of these experiments was to observe the performance of the model under different conditions, such as varying the communication range and amount of beacons present in a scenario.

Three variables were monitored during the experiments: (1) the average decay level of the scenario, (2) the average number of nodes over given thresholds, and (3) the global energy consumption of the scenario. We defined three thresholds for determining the operative capability of the nodes: $T_{>0}$ to represent nodes that were able to obtain their position at any point in the past but were then unable to do so again, $T_{>20}$ to represent nodes that are above our lower bound for participation in the positioning process, and $T_{>60}$ to represent nodes that are fit to collaborate. All results shown represent the average from the test battery, unless noted otherwise.

Figure 6-5 illustrates the average fitness of non-beacon nodes in scenarios with different proportions of this node type. Each graph portrays three curves for communication ranges of 30, 50 and 80 meters, respectively. This particular set of results was obtained in a scenario with 130 nodes.



(a) 10% of beacon nodes

(b) 25% of beacon nodes

(c) 50% of beacon nodes

Figure 6-5. Fitness distribution (averaged) of non-beacon nodes per communication range

Figure 6-5-a shows the results for a scenario with 10% of beacon nodes. The average fitness of the scenario reaches about 4.05% at a communication range of 30 meters, 11.26% at 50 meters, and 22.07% at 80 meters. The scenario with 25% beacons (Figure 6-5-b) shows slightly better results, reaching average values of 9.46%, 24.32%, and 41.89% for each of the communication ranges. As for the 50% beacon scenario (Figure 6-5-c), the fitness reaches 16.67%, 36.48%, and 54.95% for communication ranges of 30, 50 and 80 meters, respectively.

Results show that the greater the communication ranges of the devices, the greater the fitness levels of the scenarios. The increment values for fitness Figure 6-5-a is +7.21% when augmenting the range from 30 to 50 meters, and +10.81% when going from 50 to 80 meters. In a similar manner, the average fitness grows consistently as the number of

beacons present in the scenario increases, amounting to +14.86% and +17.57% in Figure 6-5-b, and +19.82% and +18.47% in Figure 6-5-c.

Similarly, as more beacons are present in the scenario, the fitness levels tend to rise. For a communication range of 30 meters, when comparing the graphs 6-5-a and 6-5-b, there is an increment of +5.40% in fitness, and of +7.21% when comparing graphs 6-5-b and 6-5-c. For a communication range of 50 meters, the fitness increment is of +13.06% and +12.16%, and for 80 meters the increment is of +19.82% and +13.06%.

However, to determine whether the proposed model is actually helping non-beacon nodes to perform positioning, we must measure how many of them actually know their positions. It is not enough to just count nodes with known positions; it is necessary to determine whether their positioning information is up-to-date or not, and thus whether they are fit for collaboration. This information can be obtained by observing their fitness scores.

Figure 6-6 shows the amount of nodes (%) that have fitness scores over the given thresholds ($T_{>0}$ and $T_{>60}$), for scenarios with communication ranges of 30, 50 and 80 meters. There are two sets of curves on each graph, all representing the number of nodes over a given fitness threshold for each of the three proportions of beacons (10%, 25%, and 50%). The solid curves represent the percentage of non-beacons with fitness scores over 60 ($T_{>60}$), while the dotted curves show the percentage of non-beacons with fitness scores greater than 0 ($T_{>0}$).

For the scenario with a communication range of 30 meters (Figure 6-6-a), the average percentage of nodes with fitness $T_{>60}$ is of 0.45 nodes when 10% beacons are present, 1.80 for 25% of beacons, and 3.60 for 50% of beacons. This means that there are basically no nodes that have fitness levels fit for collaboration. The quantity of $T_{>0}$ nodes is generally higher, with 10.81, 20.27, and 21.17 for each of the beacon proportions, respectively.

When the communication range is raised to 50 meters (Figure 6-6-b), the average percentage of nodes over threshold $T_{>60}$ increases to 4.05, 9.01, and 13.51 for each proportion of nodes. The count of $T_{>0}$ nodes is of 35.59, 48.64, and 37.84, for the 10%, 25%, and 50% beacon amounts, respectively. Finally, for a communication range of 80 meters (Figure 6-6-c), the average percentage of nodes amounts to 4.95, 19.37, and 27.93 for the $T_{>60}$ nodes, and for the $T_{>0}$ nodes it reaches values of 72.07, 68.47, and 46.84 for each of the three beacon proportions, respectively.

Results show that the number of nodes over both, the $T_{>60}$ and the $T_{>0}$ fitness thresholds, increases as the communication range of the devices is augmented, as expected. This increase is smaller when the threshold is greater (e.g., 60), but more noticeable when the threshold is lower (e.g., 20). When 10% of beacon nodes are present in a scenario, incrementing the communication range from 30 to 50 meters results in +1.35% of $T_{>60}$ nodes (Figures 6-6-a and 6-6-b), and +1.80% when going from 50 to 80 meters (Figures 6-6-b and 6-6-c). The amount of nodes with $T_{>0}$ has a similar behavior,

growing +9.46% and +0.90% when increasing the communication range from 30 to 50 meters, and from 50 to 80, respectively.

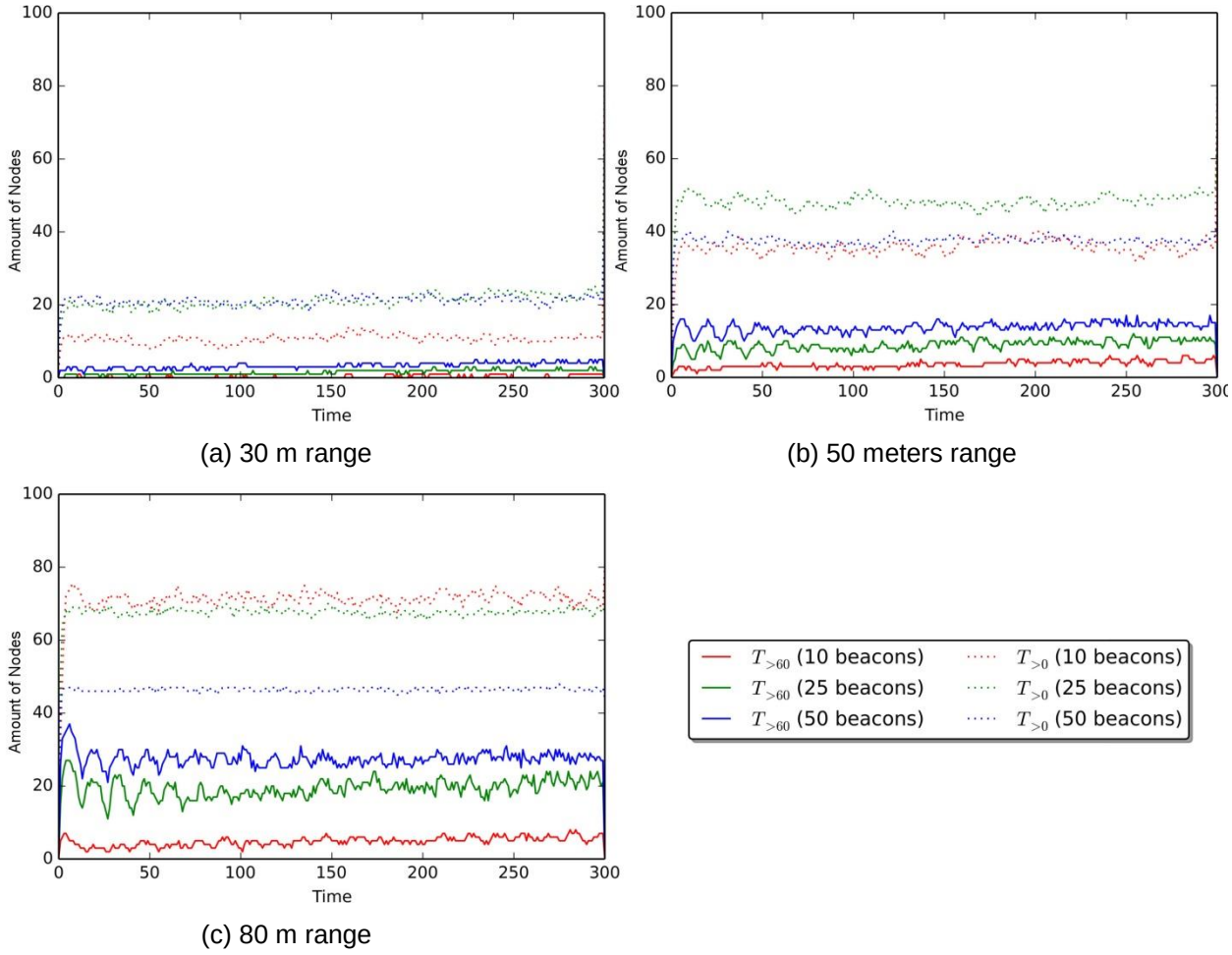


Figure 6-6. Percentage of nodes with fitness over thresholds $T_{>0}$ and $T_{>60}$, per proportion of beacons

For the 25% beacon proportion scenario, the $T_{>60}$ nodes show an increase of +4.95% and +4.50% when the communication range is incremented from 30 to 50 meters (Figures 6-6-a and 6-6-b), and then from 50 to 80 meters (Figures 6-6-b and 6-6-c), respectively. Meanwhile, $T_{>0}$ nodes show an increase of +2.25% and +10.81%, when going from a communication range of 30 to 50 meters, and from 50 to 80 meters, respectively.

Finally, for the scenario with a 50% beacon proportion, the number of $T_{>60}$ nodes increases by +14.41% and +8.56%, while the $T_{>0}$ node amount grows by +21.62% and +3.60% nodes for the corresponding increments in the communication range.

In terms of energy consumption, results are straightforward. Figure 6-7 shows three different curves that portray the average energy consumption of a set of scenarios with a proportion of 25% beacons, a communication range of 30 meters, and an amount of nodes of 130.

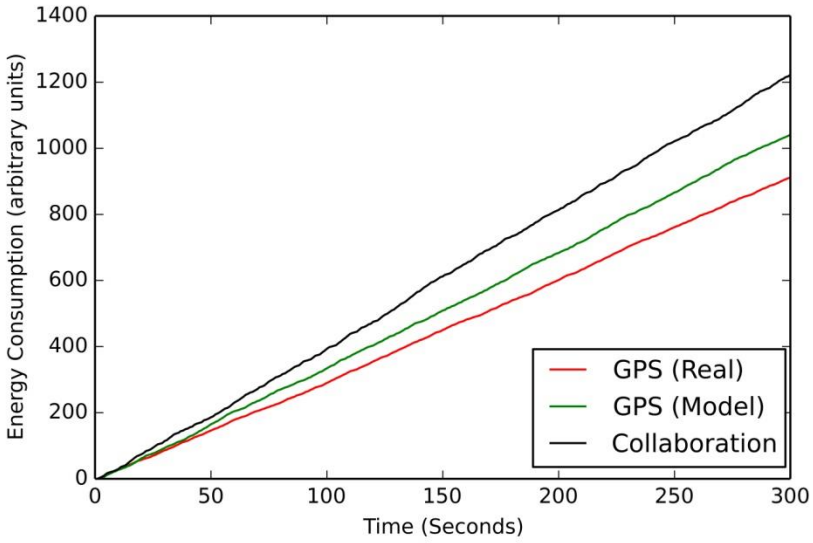


Figure 6-7. Energy consumption (standard scenario, with 130 nodes)

These results were calculated based on the Carrol et al. [28] energy consumption formula for GPS ($166 \pm 0.04\% \, mW$ per request). Results are averaged from the entire scenario, and they include three curves: (1) one for emulated self-positioning GPS (GPS Real), (2) one for self-positioning GPS under CAMPOS management (GPS Model), and (3) another for CAMPOS trilateration (collaboration).

In all cases, the energy consumption is almost perfectly linear, because every communication attempt and information exchange requires expending energy. Assuming the curves are straight lines, the slope of each curve would be determined by the quantity of energy expended; the more energy spent, the greater the slope value.

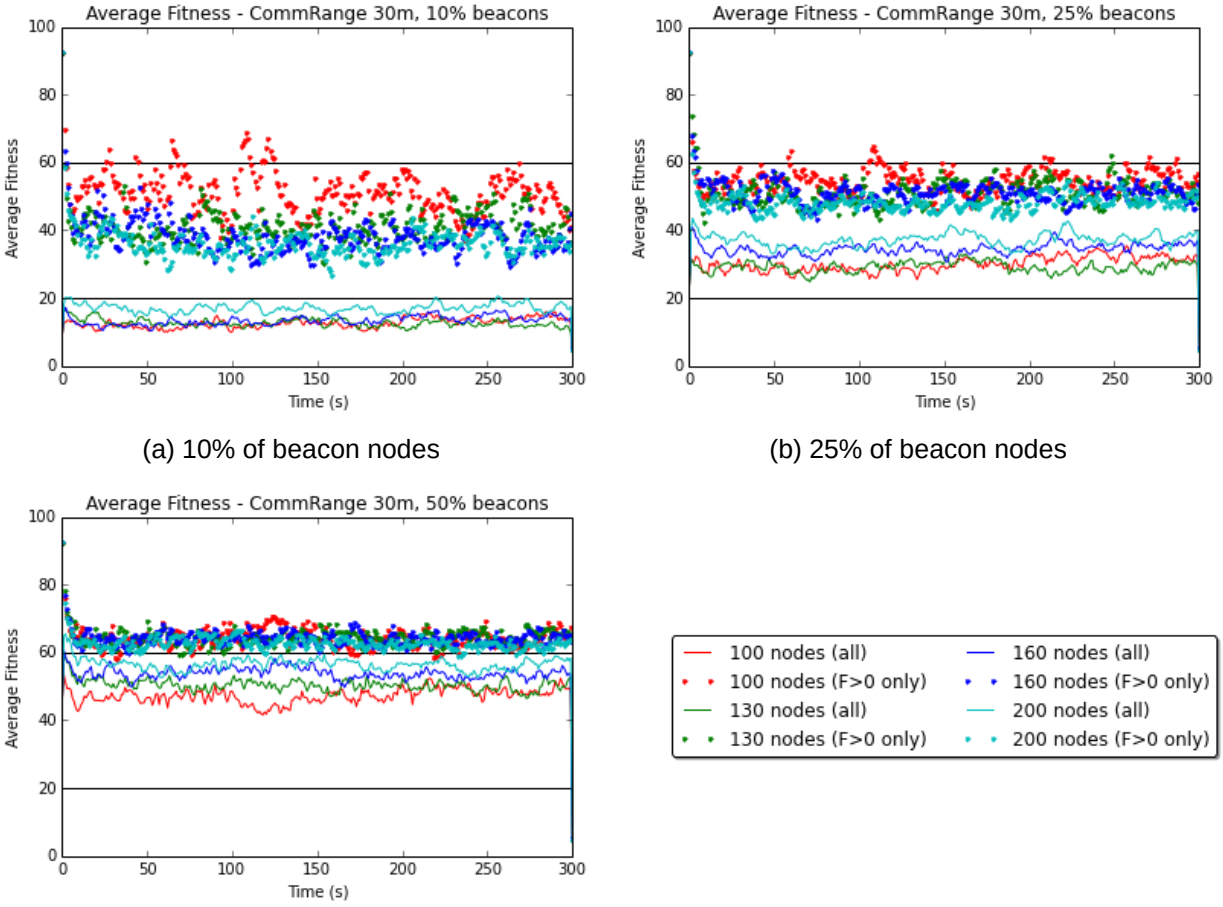
The GPS Real curve was determined using Carrol's formula, with default intervals for GPS positioning requests. The values from this curve were measured in a real world device, and then emulated inside the simulation environment. It has the lower energy consumption values from all curves in Figure 6-7, amounting to 915 units after 300 seconds of measuring.

The second curve, GPS Model, uses the same formula as GPS Real, but instead of using the default GPS positioning request rate, it uses CAMPOS decay-based positioning request scheme. At the end of the simulation, this curve has a total energy consumption of 1037 units, an increment of +122 over vanilla GPS.

Finally, the Collaboration curve, which uses CAMPOS decay-based positioning scheme, scored 1221 energy units consumed after 300 seconds. This result amounts to a difference of +306 units over GPS Real, and +184 units over GPS Model.

6.2.2. Late Model Development: Random Forest

These sets of results are from the latest iteration of the model, in which we treat the problem of determining the best alternative to provide positioning to a device, based on its context as a classification problem. As explained in Sections 4 and 5, we settled for using a random forest classifier, with the feature vectors corresponding to a device and its contextual scenario as input, and the recommended positioning strategy as output. In addition, the decay formula (2) had to be revised, adding a coefficient (k) to cope with the huge difference in decrement between vehicles and pedestrians.



(a) 10% of beacon nodes

(b) 25% of beacon nodes

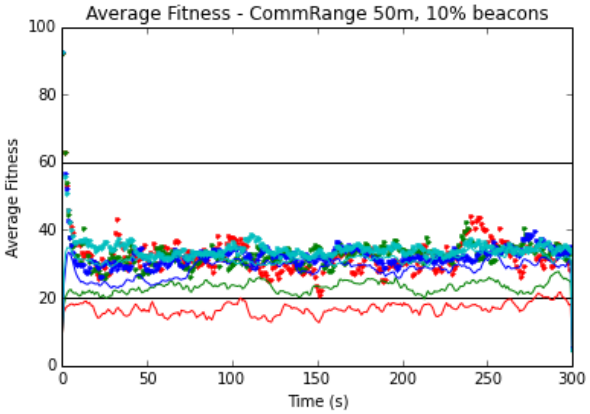
(c) 50% of beacon nodes

Figure 6-8. Average fitness per amount of nodes in scenario; communication range of 30 meters

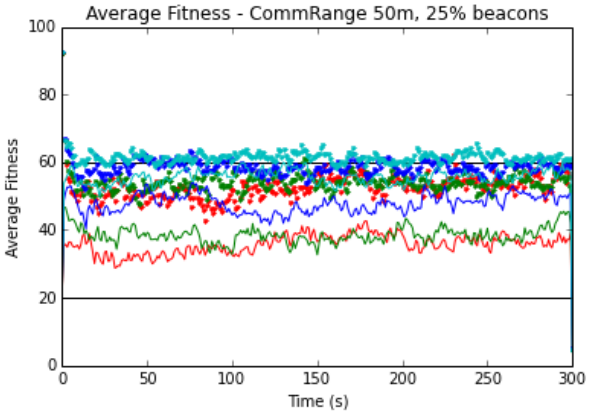
Figures 6-8, 6-9, and 6-10 depict the average fitness results of fifteen CAMPOS test simulations, grouped by communication range. Each graph shows the average fitness of non-beacon nodes for all four amounts of nodes, across the three observed beacon proportions. The solid lines represent the average results of all non-beacon nodes ("global" fitness), and the dotted lines represent only non-beacons with fitness greater than 0 ($F_{>0}$). The latter is included to provide an idea of the average fitness of the nodes that have managed to obtain their positions, avoiding bias by isolated nodes. Two

horizontal black lines have been traced, to represent the $T_{>20}$ and $T_{>60}$ thresholds.

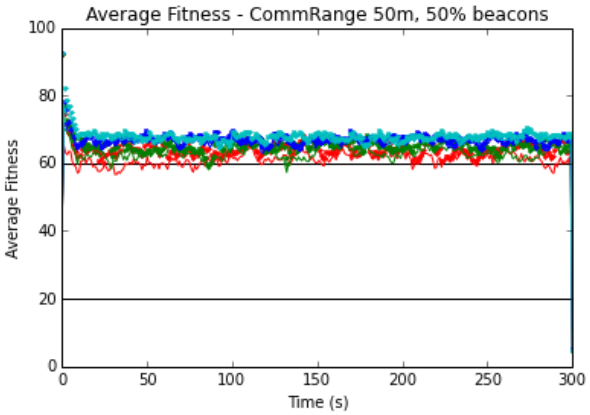
Due to the amount of data that must be shown, we will not present detailed fitness values for each curve and graph to avoid cluttering this subsection. However, we will provide trends and values for specific scenarios instead.



(a) 10% of beacon nodes



(b) 25% of beacon nodes



(a) 50% of beacon nodes

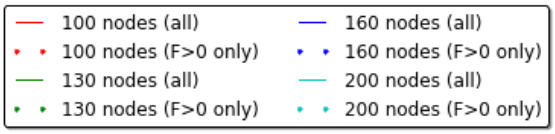


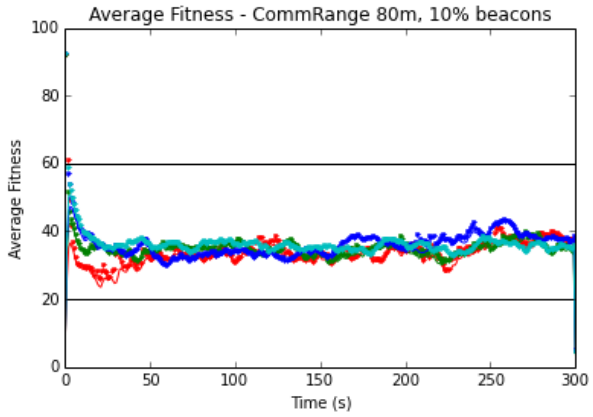
Figure 6-9. Average fitness per amount of nodes in scenario; communication range of 50 meters

The results shown in Figure 6-8 correspond to a communication range of 30 meters. The global fitness values for a beacon proportion of 10% (Figure 6-8-a) remain consistently above 10 units, but do not reach minimum fitness operating levels. For all amounts of nodes, the fitness is within a range of $[10, 20]$. However, when nodes with a fitness of zero are excluded from the average ($F_{>0}$), the fitness rises, reaching values of 47.87 to up to 56.74 (above $T_{>20}$, and thus within operating levels).

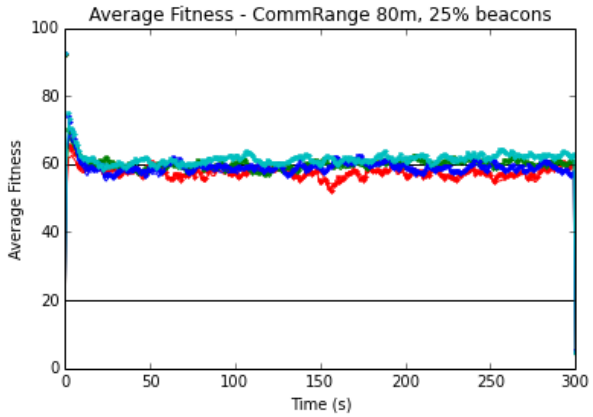
When incrementing the proportion of beacons to 25% (Figure 6-8-b), global fitness rises above 25, but below 40. On the other hand, values for $F_{>0}$ nodes rise to within a range of 40 and 60. For a proportion of the 50% of beacons (Figure 6-8-c), the ranges of fitness increase to within 40 and 60 units for the global average, and above 60 for the $F_{>0}$

nodes. The latter values are within acceptable levels for our model, although that density of beacons in such a small scenario is unlikely.

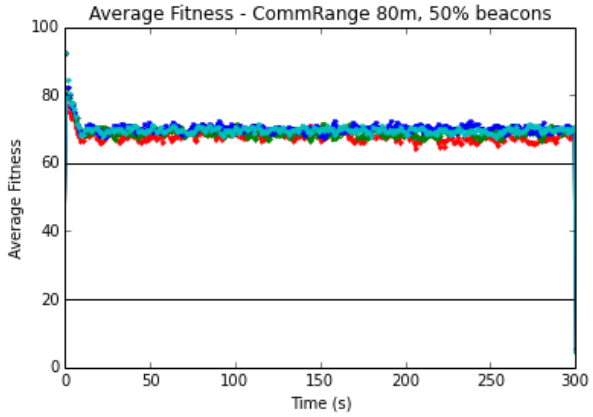
For a communication range of 50 meters (Figure 6-9), the trend is similar, with fitness growing as the communication range and proportion of beacons increases. The ranges for global average fitness are of $[15, 30]$, $[30, 50]$, and $[60, 70]$ for the 10%, 25% and 50% proportions of beacons, respectively. Conversely, the values for $F_{>0}$ nodes are within ranges of $[25, 40]$, $[50, 65]$, and $[60, 70]$, for the same proportions of beacons.



(a) 10% of beacon nodes



(b) 25% of beacon nodes



(a) 50% of beacon nodes

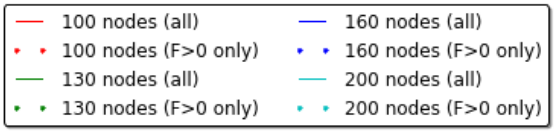


Figure 6-10. Average fitness per amount of nodes in scenario; communication range of 80 meters

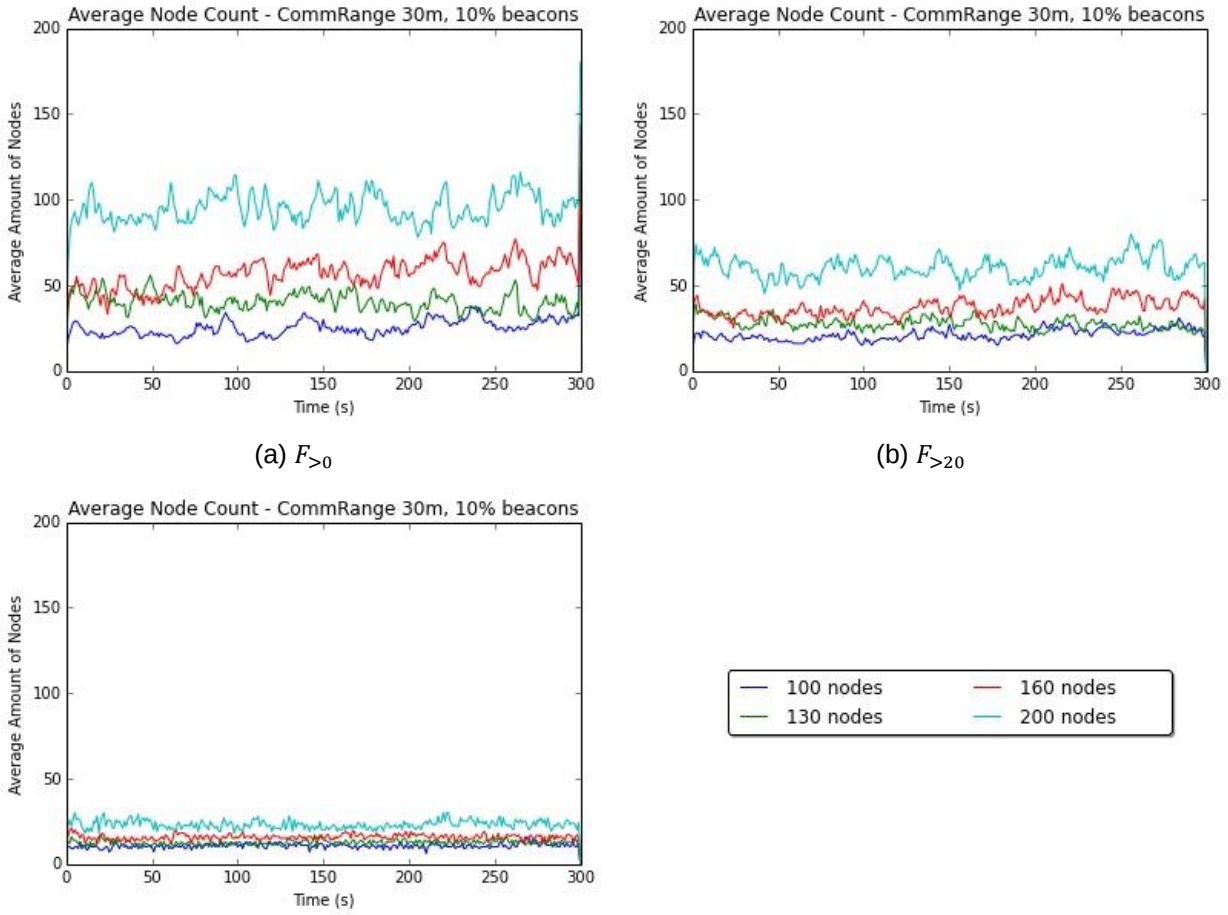
Finally, for the 80 meters communication range (Figure 6-10), both the global fitness and the fitness of $F_{>0}$ nodes for a 10% of beacon proportion are within a range of $[60, 70]$. Similarly, for the 25% of beacon proportion, both the global and $F_{>0}$ fitness are within a range of $[55, 65]$, and within $[65, 75]$ for a 50% of beacon proportion.

Regarding the amounts of nodes with known positions, Figures 6-11 through 6-19 show the amount of nodes that know their position with fitness over the three defined thresholds, $F_{>0}$, $F_{>20}$ and $F_{>60}$. These results take into account both beacon and non-

beacon nodes, to avoid confusion due to the graphs being based on the “amount of nodes in scenario” dimension (not percentages, but actual amounts).

Since there is a significant amount of graphs depicting the behavior of the amount of nodes given a certain threshold (a total of 27), we decided to group them in three subsections. Each subsection deals with a specific communication range, and includes figures with the average results for each of the beacon proportions. For a summary and a detailed discussion of these results, please refer to Section 6.3.2, specifically Tables 6-8 and 6-9.

6.2.2.1. Node count in scenarios with a communication range of 30 meters



(a) $F_{>0}$

(b) $F_{>20}$

(c) $F_{>60}$

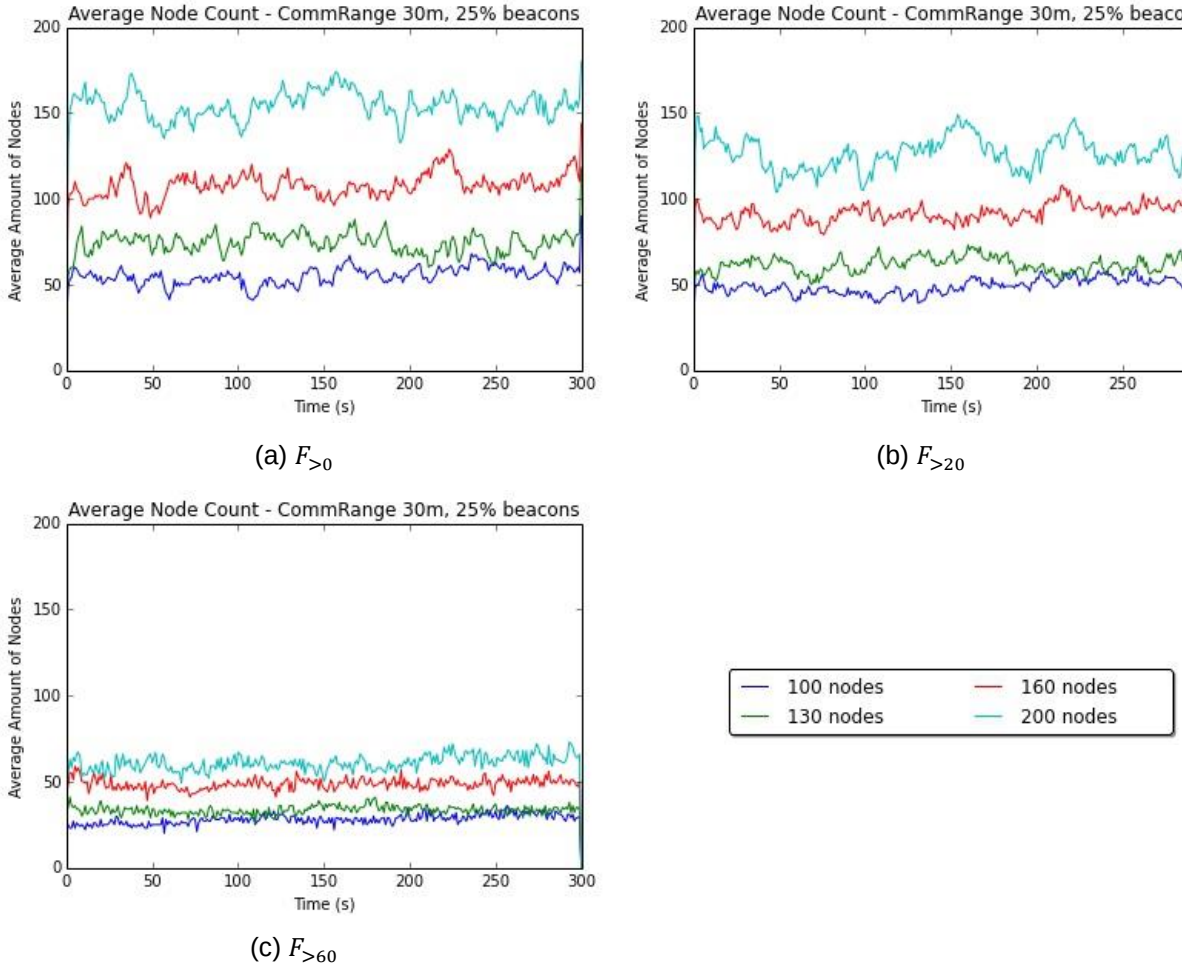
Figure 6-11. Amount of nodes with fitness over threshold (including beacons) per fitness threshold; communication range of 30 meters, 10% of beacons

The scenario with a communication range of 30 meters and a proportion of 10% beacons (Figure 6-11) shows that for the 100 node scenario, only an average of 28 nodes knows their position with any degree of fitness ($F_{>0}$), 23 have operating fitness values ($F_{>20}$), and barely 11 have adequate fitness ($F_{>60}$). This is the worst case scenario, in which only one non-beacon has a fitness value adequate to be a reference point for its

neighbors (subtracting the 10 beacons of the scenario, we get only one $F_{>60}$ node).

For the 130 node scenario, an average of 46 nodes have $F_{>0}$, 38 have $F_{>20}$, and 13 have $F_{>60}$ fitness. The 160 node scenario has 57 $F_{>0}$ nodes, 46 $F_{>20}$ nodes, and 17 $F_{>60}$ nodes. Finally, the 200 node scenario has 98, 61, 22 nodes with fitness over thresholds of $F_{>0}$, $F_{>20}$, and $F_{>60}$, respectively.

Figure 6-12 shows the results for the scenario with a communication range of 30 meters and a proportion of 25% of beacons. The 100 node scenario shows an average of 53 $F_{>0}$ nodes, 51 $F_{>20}$, and 27 $F_{>60}$. For the 130 node scenario, there are 69, 55, and 37 nodes for the $F_{>0}$, $F_{>20}$, and $F_{>60}$ nodes, respectively.



(a) $F_{>0}$

(b) $F_{>20}$

(c) $F_{>60}$

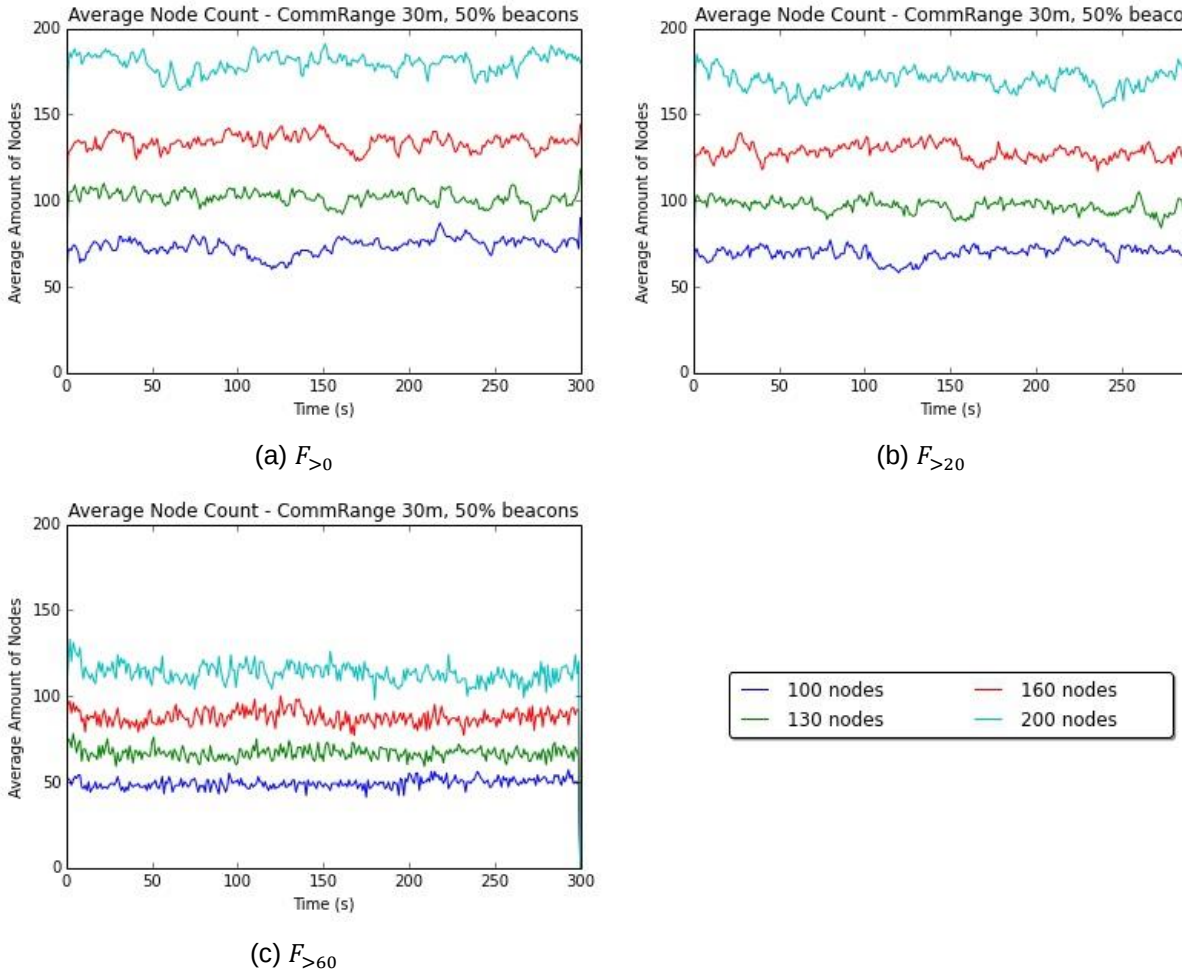
Figure 6-12. Amount of nodes with fitness over threshold (including beacons) per fitness threshold; communication range of 30 meters, 25% beacons

The 160 node scenario shows 108 $F_{>0}$ nodes, 97 $F_{>20}$, and 50 $F_{>60}$. Finally, the 200 node scenario has nodes in amounts of 154 for the $F_{>0}$ threshold, 128 for $F_{>20}$, and 56 for the $F_{>60}$ threshold.

The 130 node scenario for this communication range and beacon proportion (30

meters and 25%, respectively) is the first scenario in which the model provides over 50% of nodes with known positions and fitness within operating ($F_{>20}$).

The final scenario for the communication range of 30 meters is the 50% beacon node proportion, portrayed in Figure 6-13, which shows results of 66, 64 and 52 nodes for the thresholds of $F_{>0}$, $F_{>20}$, and $F_{>60}$ for the 100 node scenario, respectively.



(a) $F_{>0}$

(b) $F_{>20}$

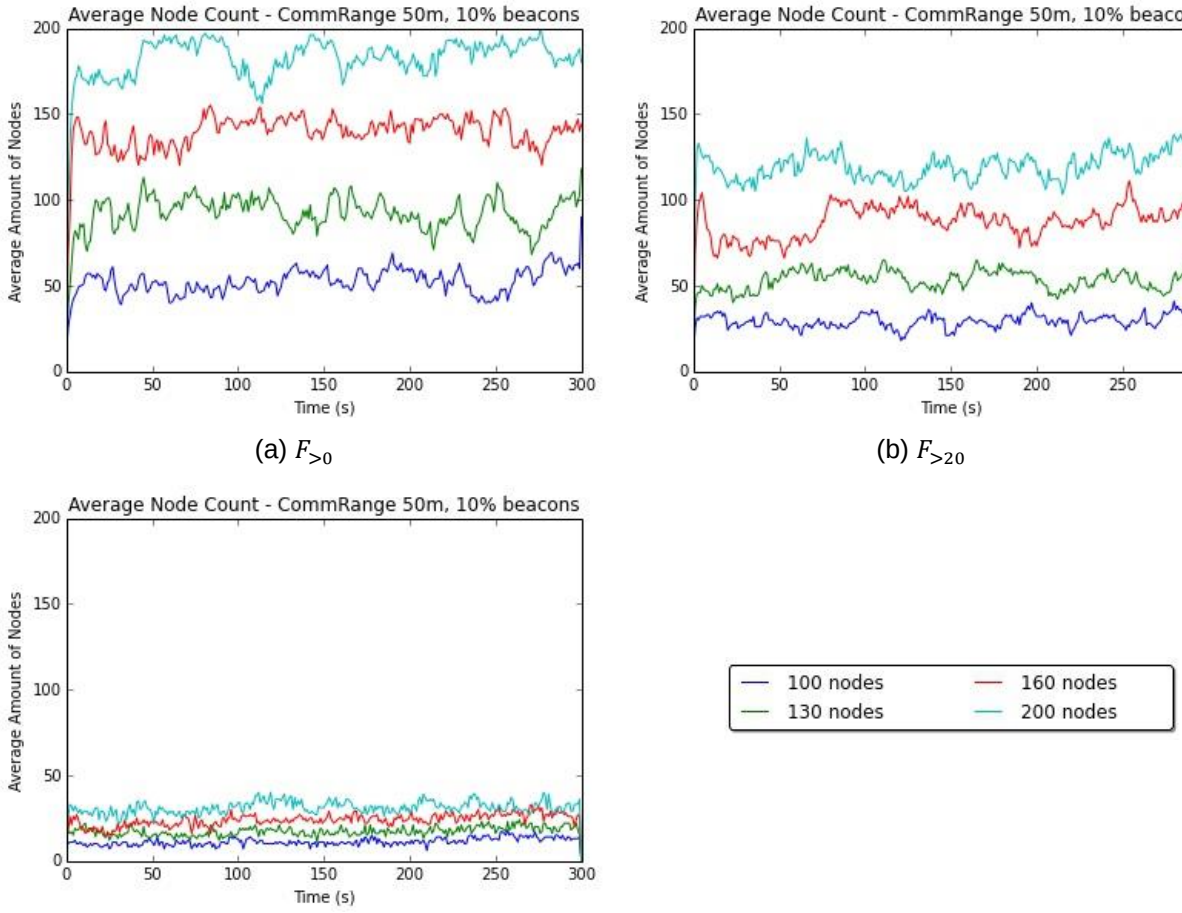
(c) $F_{>60}$

Figure 6-13. Amount of nodes with fitness over threshold (including beacons) per fitness threshold; communication range of 30 meters, 50% beacons

These numbers change to 101, 97, and 71 for the 130 node scenario; 129, 126, and 95 for the 160 node scenario; and 183, 176, 113 nodes over the established thresholds for the 200 node scenario.

6.2.2.2. Node count in scenarios with a communication range of 50 meters

This subsection groups the results for the scenarios with a communication range of 50 meters, across all three beacon proportions. Figure 6-14 shows the average results of scenarios with a proportion of beacons of 10%. For the scenarios with 100 nodes, there are a total of 51 nodes with fitness over threshold $\mathrm{F}_{>0}$, 27 nodes over $\mathrm{F}_{>20}$and 13 nodes over $\mathrm{F}_{>60}$, in average.



(a) $F_{>0}$

(b) $F_{>20}$

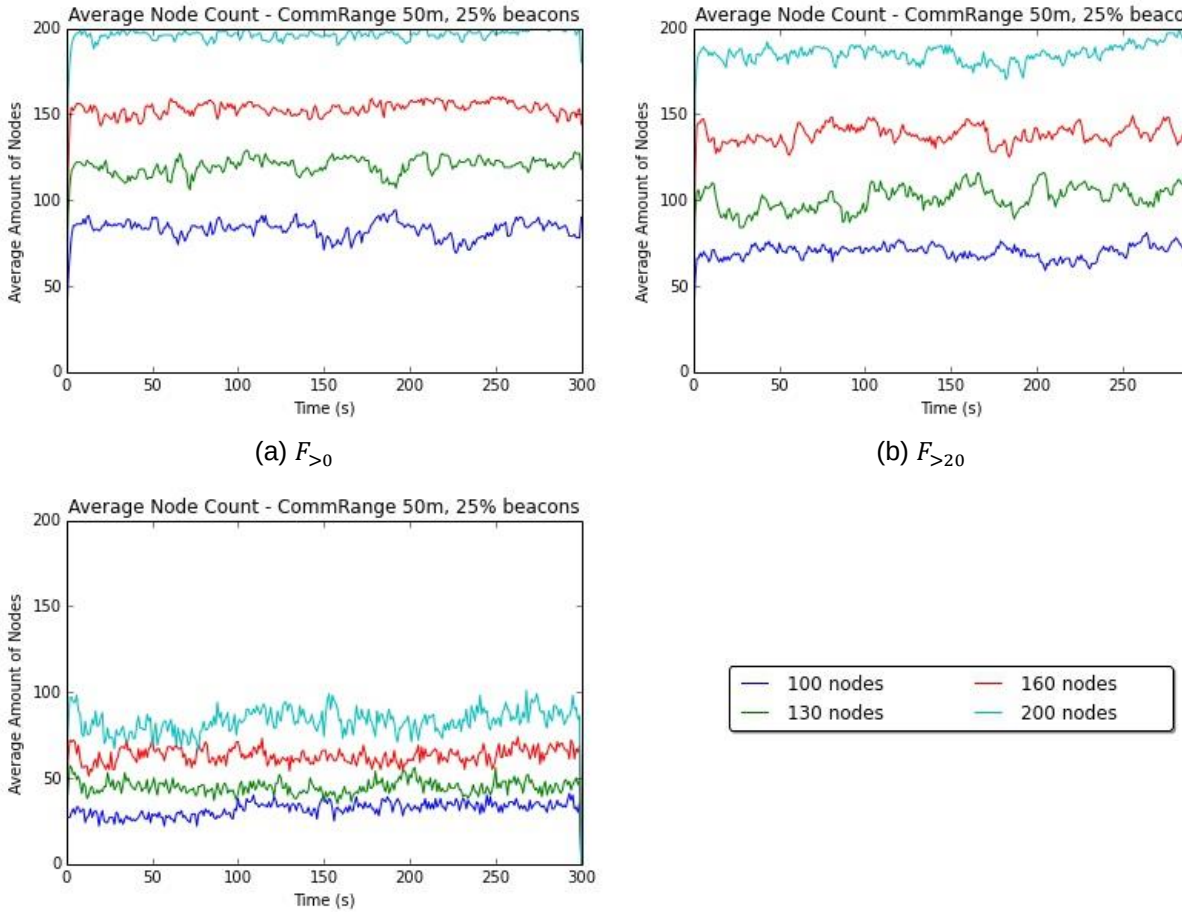
(c) $F_{>60}$

Figure 6-14. Amount of nodes with fitness over threshold (including beacons) per fitness threshold; communication range of 50 meters, 10% beacons

When the number of nodes in the scenario increases to 130, the nodes with fitness over threshold increases to 83, 51, and 17 for the $F_{>0}$, $F_{>20}$and $F_{>60}$, respectively. These numbers increase again to 144 nodes over fitness threshold $F_{>0}$, 89 over $F_{>20}$, and 24 nodes over $F_{>60}$, when the amount of nodes in the scenario is set to 160.

When the number of nodes in the scenario amounts to 200, the count of nodes with fitness over threshold increases to 178 for the $F_{>0}$ threshold, 124 for the $F_{>20}$ threshold, and 33 for the $F_{>60}$ threshold, respectively.

For the beacon proportion of 25%, shown in Figure 6-15, the scenario with 100 nodes shows 77 nodes with fitness $F_{>0}$, 62 with fitness $F_{>20}$, and 27 with fitness $F_{>60}$, on average. The scenario with a population of 130 nodes shows results amounting to 117 nodes with $F_{>0}$, 102 nodes with $F_{>20}$, and 48 nodes with $F_{>60}$, across the three graphs.



(a) $F_{>0}$

(b) $F_{>20}$

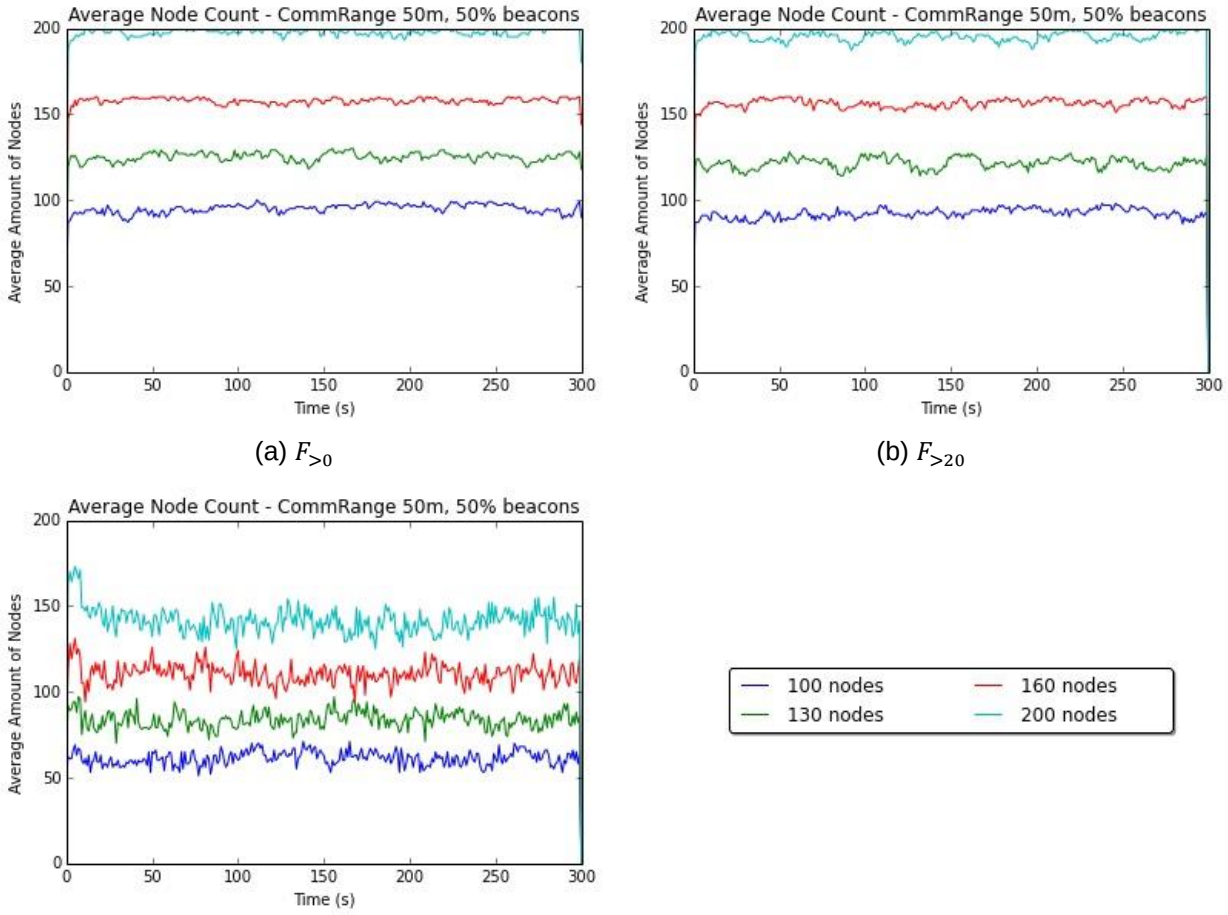
(c) $F_{>60}$

Figure 6-15. Amount of nodes with fitness over threshold (including beacons) per fitness threshold; communication range of 50 meters, 25% beacons

The scenario with 160 nodes shows results of 151 nodes with fitness $\mathrm{F}_{>0}$, 139 with fitness $\mathrm{F}_{>20}$, and 61 with fitness $\mathrm{F}_{>60}$. Finally, the 200 node scenario shows results of 196, 188, and 83 fitness, for each of the three fitness thresholds $\mathrm{F}_{>0}$, $\mathrm{F}_{>20}$, and $\mathrm{F}_{>60}$.

Figure 6-16 shows results for the 50% proportion of beacons. When 100 nodes are present in the scenario, the amount of nodes over the $\mathrm{F}_{>0}$ threshold amount to 95, 93 for the $\mathrm{F}_{>20}$ threshold, and 56 for the $\mathrm{F}_{>60}$ threshold. For an amount of 130 nodes in a scenario, the node count with fitness over $\mathrm{F}_{>0}$ reaches 122, and it decreases to 119 for the $\mathrm{F}_{>20}$, and to 87 for the $\mathrm{F}_{>60}$.

For the scenario with 160 nodes, there are 154 nodes with fitness $\mathrm{F}_{>0}$, 153 nodes with $\mathrm{F}_{>20}$, and 121 nodes with $\mathrm{F}_{>60}$. These values increase to 197 for the $\mathrm{F}_{>0}$ threshold, 194 for $\mathrm{F}_{>20}$, and 145 for $\mathrm{F}_{>20}$ in scenarios with 200 nodes.



(a) $F_{>0}$

(b) $F_{>20}$

(c) $F_{>60}$

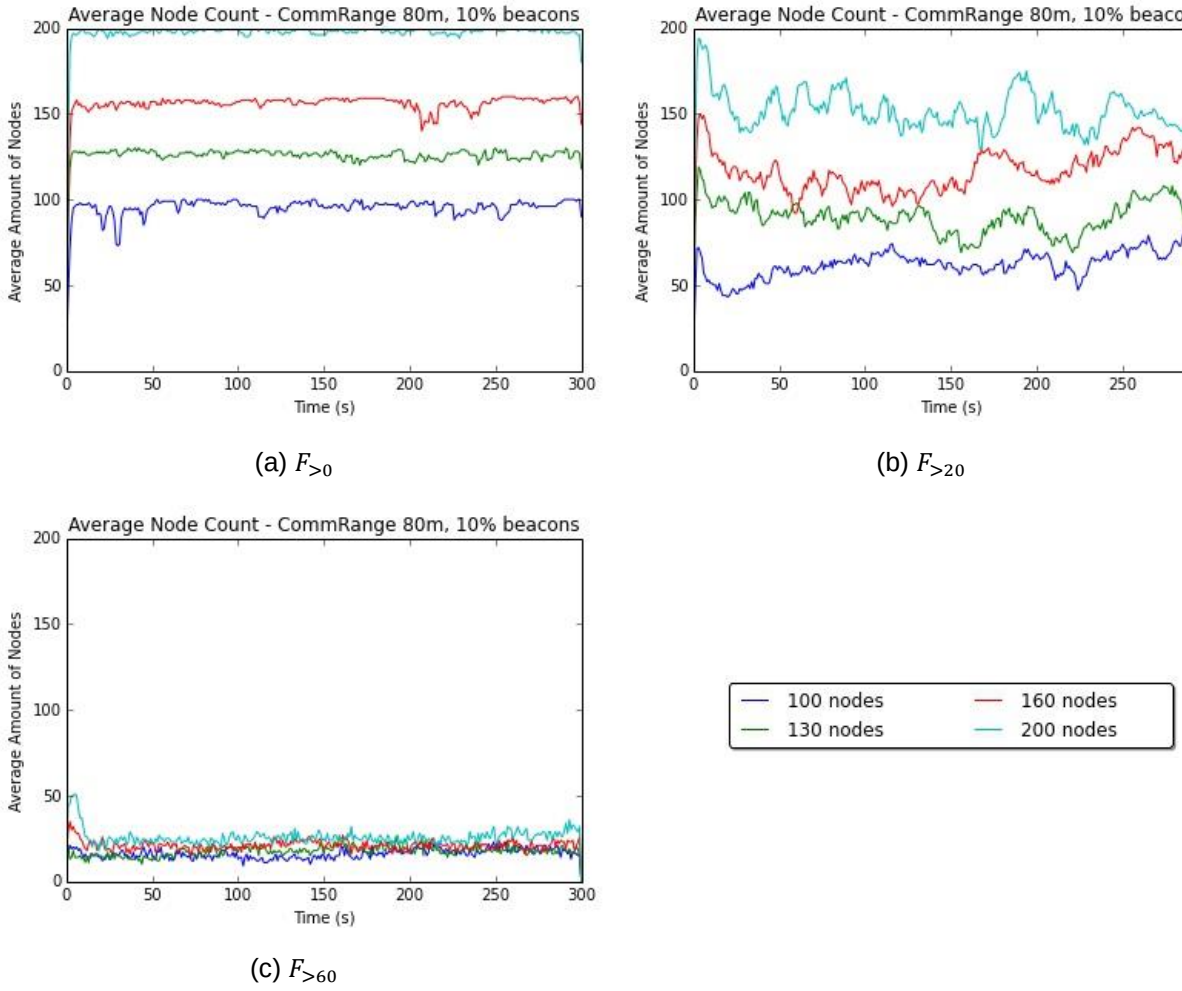
Figure 6-16. Amount of nodes with fitness over threshold (including beacons) per fitness threshold; communication range of 50 meters, 50% beacons

6.2.2.3. Node count in scenarios with a communication range of 80 meters

This subsection deals with the results from the scenarios with communication ranges of 80 meters, for each of the three proportions of beacons. Figure 6-17 shows results of 93 nodes for the $F_{>0}$ fitness threshold, 62 nodes for the $F_{>20}$ threshold, and 17 for the $F_{>60}$ threshold.

In the scenario with 130 nodes, the number of nodes over a given threshold are of 126 for the $F_{>0}$ threshold, 97 for the $F_{>20}$ threshold, and 17 for the $F_{>60}$ threshold. For the scenario with a population of 160 nodes, the number of $F_{>0}$ nodes is 151, a number that decreases for $F_{>20}$, reaching 128 nodes, decreasing further to 23 for the $F_{>0}$ threshold.

For the scenario of 200 nodes, the amount of nodes with fitness over the $F_{>0}$ threshold amounts to 197. This number decreases to 153 when the fitness threshold is increased to $F_{>20}$, and to 28 when the fitness threshold is further increased to $F_{>60}$.



(a) $F_{>0}$

(b) $F_{>20}$

(c) $F_{>60}$

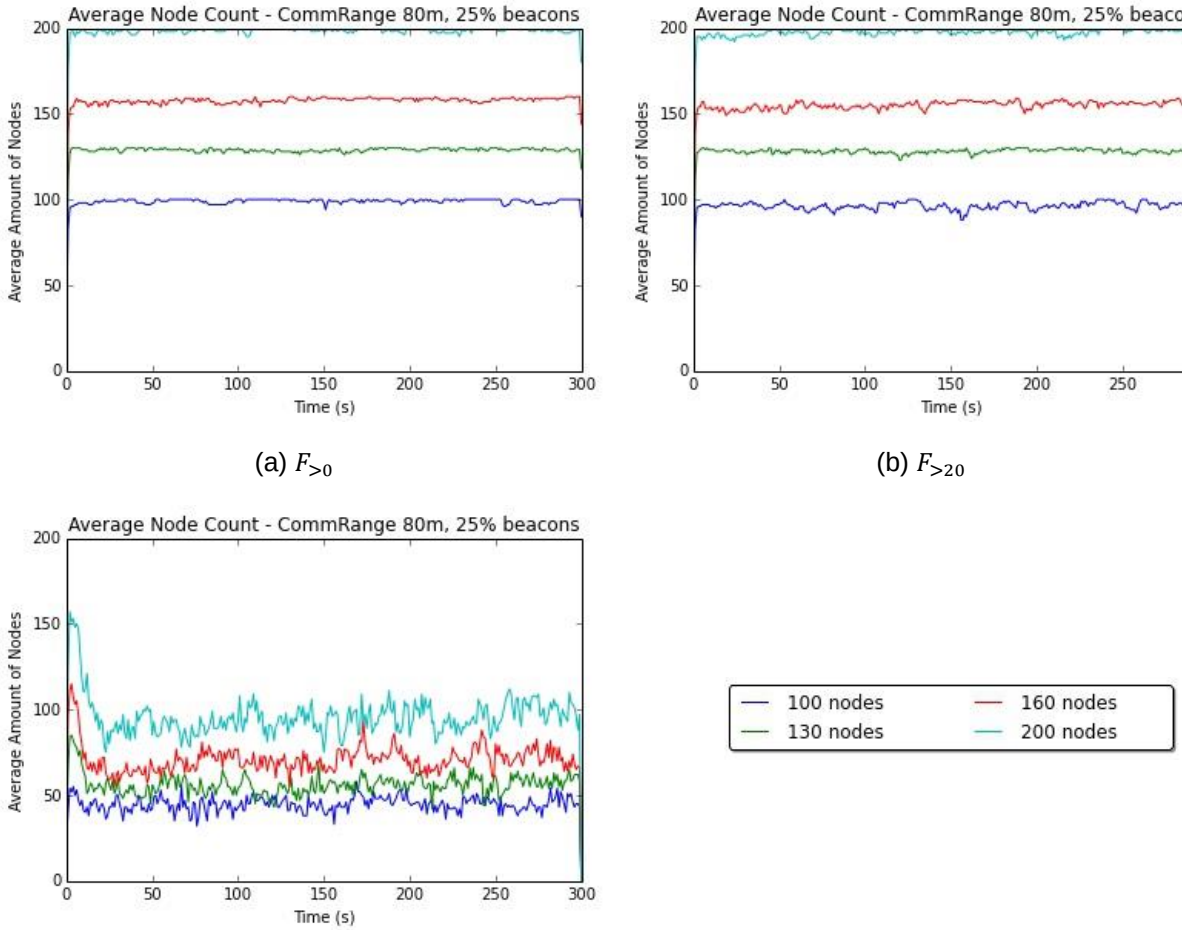
Figure 6-17. Amount of nodes with fitness over threshold (including beacons) per fitness threshold; communication range of 80 meters, 10% beacons

Figure 6-18 shows the amount of nodes for the 25% beacon proportion scenarios. For a population of 100 nodes, the amounts of nodes over a threshold are 99, 98, and 47, for the fitness thresholds $F_{>0}$, $F_{>20}$, and $F_{>60}$, respectively.

If the population is increased to 130, the count of nodes over a given fitness threshold amounts to 128, 127, and 59 nodes with fitness over the established thresholds of $F_{>0}$, $F_{>20}$, and $F_{>60}$, respectively.

When the amount of nodes in the scenario increases to 160, the amount of nodes over the $F_{>0}$ threshold is of 155, 153 for $F_{>20}$, and 71 for the $F_{>60}$. Finally, for scenarios with populations of 200 nodes, the amount of nodes with fitness over the $F_{>0}$ threshold is of 199, 198 for the $F_{>20}$ threshold, and 97 for the $F_{>60}$ threshold.

The results for the amount of nodes in scenarios with a 50% beacon proportion can be seen in Figure 6-19. For this particular communication range, almost 100% of the nodes across all population counts have operating fitness levels of above 20.



(a) $F_{>0}$

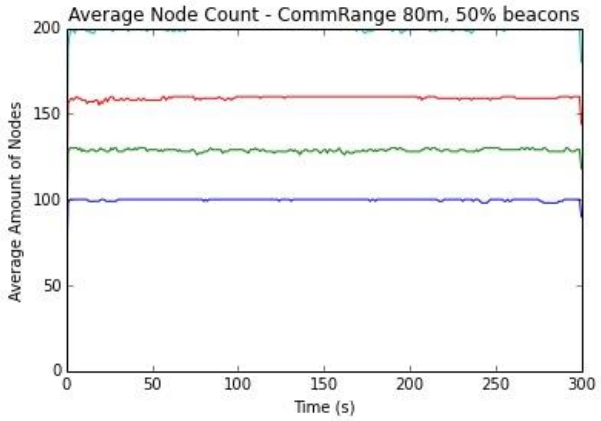
(b) $F_{>20}$

(c) $F_{>60}$

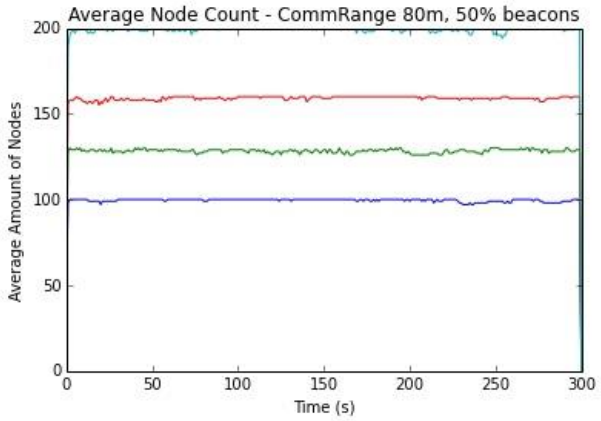
Figure 6-18. Amount of nodes with fitness over threshold (including beacons) per fitness threshold; communication range of 80 meters, 25% beacons

In the scenario with a population of 100 nodes, there is an average of 99 $\mathrm{F}_{>0}$ nodes, 99 $\mathrm{F}_{>20}$, and 81 $\mathrm{F}_{>60}$ nodes. These numbers increase to 129, 128, and 101 for the $\mathrm{F}_{>0}$, $\mathrm{F}_{>20}$, and $\mathrm{F}_{>60}$ fitness thresholds, respectively, when the population of nodes is increased to 130.

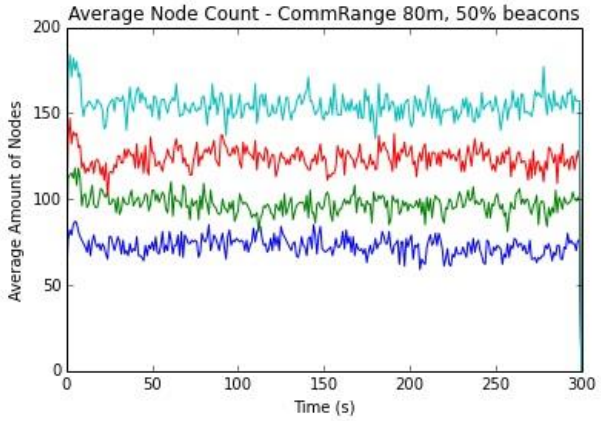
For the 160 nodes scenario, the amount of nodes with fitness values of $\mathrm{F}_{>0}$ is of 160, the amount of $\mathrm{F}_{>20}$ nodes is of 159, and that of $\mathrm{F}_{>60}$ nodes reaches 126. Finally, for the 200 nodes scenarios, the fitness thresholds of $\mathrm{F}_{>0}$, $\mathrm{F}_{>20}$, and $\mathrm{F}_{>60}$, have values of 200, 200, and 154, respectively.



(a) $F_{>0}$



(b) $F_{>20}$



(c) $F_{>60}$

Figure 6-19. Amount of nodes with fitness over threshold (including beacons) per fitness threshold; communication range of 80 meters, 50% beacons

6.3. Analysis and Discussion

In this subsection, we provide an analysis of the results shown in Section 6.2. For each of the two experiments, we provide a discussion on the reasons behind the obtained results, as well as explanations on our findings and the consequences or benefits they carry.

6.3.1. Early Model Development: Rule-based Heuristics

The most notable finding was that the model does provide positioning to non-beacon nodes. This fact is illustrated in Tables 6-5 and 6-6, which show the average percentage of nodes that know their position with fitness over certain thresholds. Table 6-5 shows the amounts of nodes that have any value for fitness ($T_{>0}$), and Table 6-6 shows the amounts of nodes for fitness ($T_{>60}$). Remember that fitness determines the “confidence” on the node’s positioning information, and it is an indicator that the node is a valid reference point during the positioning process.

Table 6-5: Percentage of nodes with fitness over thresholds $T_{>0}$, per proportion of beacons.

Communication Range (m)	Beacon Proportion		
	10%	**25%**	**50%**
30	11%	20%	21%
50	36%	49%	38%
80	72%	68%	47%

For instance, let us assume a scenario populated by nodes with a proportion of 25% beacons, and a communication range of 50 meters. In such a scenario, there are 49% non-beacons that have managed to determine their positions at any point during the simulation ($T_{>0}$, Table 6-5), for a grand total of 74% nodes with known positions (adding up 49% non-beacons with known positions, plus 25% beacons).

However, the amount of nodes that have meaningful positioning information in this scenario (i.e., nodes with $T_{>60}$, Table 6-6) is of only 9%, amounting up to 34 nodes with meaningful known positions.

Table 6-6: Percentage of nodes with fitness over threshold $T_{>60}$ per proportion of beacons.

Communication Range (m)	Beacon Proportion		
	10%	**25%**	**50%**
30	0%	2%	4%
50	4%	9%	14%
80	5%	19%	28%

Moreover, if the communication range increases, the population of $T_{>0}$ nodes skyrockets, reaching total percentages of 82%, 93% and 97% of nodes with known positions for each of the proportion of beacons (counting non-beacons and beacons). However, this is not the case for $T_{>60}$ nodes, which only reach percentages of 15%, 44%, and 78% of nodes with meaningful fitness.

This is explained by the fact that having more beacon nodes increases the likelihood of non-beacons finding neighbors viable for collaboration. This allows more non-beacons to estimate their positions, both increasing the fitness scores in the scenario. In addition, it allows these nodes to become reference points for other non-beacons, helping increase the amount of nodes with known positions in the scenario.

As for fitness, illustrated in Table 6-7, the overall average values remain low across all beacon proportions and communication ranges. This is explained by the fact that there are far more $T_{>0}$ nodes than $T_{>60}$, which lowers the total amount of fitness.

Table 6-7: Fitness distribution based on beacon proportion and communication range.

Beacon Proportion	Communication Range		
	30 m	50 m	80 m
10%	4	11	22
25%	9	24	41
50%	17	36	55

The standard scenario, with a proportion of 25% beacon nodes and a communication range of 50 meters, yields only an average fitness of 24, just over the minimum operating threshold of the model. These results were not sufficient to fulfill our goals, and thus, indicated that the model required improvement to be actually useful.

Related to energy consumption, the proportion of beacons proved to be counter-intuitive in terms of the obtained results. We expected that more beacon nodes would result in greater (average) energy consumption, due to additional communications to GPS satellites (no peripheral utilization was emulated, only GPS communication). However, that was not the case. Although beacon nodes tend to utilize more energy than non-beacons during the positioning estimation process, their own consumption is offset by non-beacons, which utilize less energy as more beacons are added to the scenario. This is due to having more reference points available, which translates into less positioning requests necessary to obtain meaningful positioning information during collaboration.

On the other hand, the energy consumption in terms of the communication range performed as predicted, although not with the magnitude we expected. Higher communication ranges allowed non-beacons to reach more neighbors when requesting positioning information, increasing the number of nodes that could be reached on each communication attempt. This, in turn, increased the energy consumption of the participants due to more broadcasts being heard than with lower ranges. However, this consumption was offset by the lower amount of positioning requests of non-beacons, which could find viable neighbors more easily, and thus estimate their position in fewer attempts.

Conversely, lower communication ranges do not diminish the energy consumption, but actually increase it. This is because fewer viable neighbors are reached by non-beacons when requesting collaboration. Since non-beacons are less likely to estimate their positions, and isolated individual nodes are unable to reach viable neighbors, they must expend more energy sending additional positioning requests.

Finally, limiting the communication of devices to a single channel (such as, WiFi) is less energy-demanding than allowing multiple channels. When using a single communication channel, only one positioning request has to be sent on each attempt; when using multiple channels, a request has to be sent from each of the corresponding peripherals (one per available channel). Additionally, sending requests through multiple channels implies that some requests will be received more than once by some neighbors, and others will be lost if no neighbors are capable to receive them, which also translates

into greater energy consumption.

6.3.2. Late Model Development: Random Forest

Thanks to the improvements made to the fitness formula, as well as other improvements within the contextual management component of the model, the overall fitness of all scenarios greatly improved in relation to results from previous experiments. These results, illustrated in Table 6-8, show that the latest iteration of CAMPOS reaches operating values ($F_{>20}$) even within scenarios with adverse conditions.

Table 6-8: Fitness distribution of non-beacons across all types of scenarios.

Amount of Nodes in Scenario	Comm. Range	Beacon Proportion					
		10%		25%		50%	
		All	$F_{>0}$	All	$F_{>0}$	All	$F_{>0}$
100	30 m	13	49	30	54	47	64
	50 m	18	31	36	51	61	63
	80 m	36	36	57	57	66	66
130	30 m	13	41	29	50	52	63
	50 m	22	32	40	54	63	64
	80 m	38	38	60	60	67	67
160	30 m	12	39	36	56	54	63
	50 m	27	31	47	58	65	65
	80 m	38	38	60	60	70	70
200	30 m	17	36	39	47	57	62
	50 m	31	34	54	62	67	68
	80 m	38	38	61	61	69	69

We made a differentiation between *all* and only $F_{>0}$ nodes to emphasize the variation between the fitness values from the entire non-beacon population (counting nodes that have zero fitness), and those from non-beacons that know their positions with any degree of fitness.

In general, the fitness values for nodes that have been able to determine their positions are greater than those of the entire scenario, especially when the communication range and proportion of beacons are lower. For instance, for scenarios with communication ranges of 30 meters and a beacon proportion of 10%, the scenario fitness is of 13 for populations of 100 and 130 nodes, 12 for populations of 160 nodes, and 17 for 200 nodes. Conversely, the $F_{>0}$ nodes amount to 49, 41, 39, and 36 units for the same population amounts.

This particular scenario presented a counter-intuitive trend, in which the fitness values decreased instead of increasing, as the scenario population augmented. Since there is a correlation between values for both *all* and $F_{>0}$ nodes, we attribute this trend to energy saving procedures that limit the amount of failed positioning requests before putting the node "on hold", which tends to activate when nodes get isolated.

As the communication range and the proportion of beacons increases, the values for the entire scenario and $F_{>0}$ nodes tend to close in, reaching similar values as the population of nodes in the scenario grows. For instance, the (rounded) fitness of scenarios with communication ranges of 80 meters and proportions of 50% beacons is almost identical for *all* and $F_{>0}$.

Table 6-9: Amount of nodes with fitness over thresholds, across all types of scenarios.

Amount of Nodes in Scenario	Comm. Range	Beacon Proportion								
		10%			25%			50%		
		$F_{>60}$	$F_{>20}$	$F_{>0}$	$F_{>60}$	$F_{>20}$	$F_{>0}$	$F_{>60}$	$F_{>20}$	$F_{>0}$
100	30 m	11	23	28	27	51	53	52	64	66
	50 m	13	27	51	27	62	77	56	93	95
	80 m	17	62	93	47	98	99	81	99	99
130	30 m	13	38	46	37	55	69	71	97	101
	50 m	17	51	83	48	102	117	87	119	122
	80 m	17	97	126	59	127	128	101	128	129
160	30 m	17	46	57	50	97	108	95	126	129
	50 m	24	89	144	61	139	151	121	153	154
	80 m	23	128	151	71	153	155	126	159	160
200	30 m	22	61	98	56	128	154	113	176	183
	50 m	33	124	178	83	188	196	145	194	197
	80 m	28	153	197	97	198	199	154	200	200

Tables 6-9 and 6-10 show the amount of nodes that know their positions with any degree of fitness; the first in absolute values, and the latter as percentages. Note that values provided in these tables are cumulative for each fitness threshold, i.e., $F_{>0}$ contains both $F_{>20}$ and $F_{>60}$, and $F_{>20}$ contains $F_{>60}$. This means that the 90% node percentage of Table 6-10, for the communication range of 50 meters, a proportion of beacons of 25%, and population of 130, breaks out into three values: 37% $F_{>60}$, 41% $F_{<20,60>}$, and 12% $F_{[0,20>}$.

Table 6-10: Percentage of nodes with fitness over thresholds, across all scenarios.

Amount of Nodes in Scenario	Comm. Range	Beacon Proportion								
		10%			25%			50%		
		$F_{>60}$	$F_{>20}$	$F_{>0}$	$F_{>60}$	$F_{>20}$	$F_{>0}$	$F_{>60}$	$F_{>20}$	$F_{>0}$
100	**30 m**	10%	29%	35%	28%	42%	53%	55%	75%	78%
	50 m	13%	39%	64%	37%	78%	90%	67%	92%	94%
	80 m	13%	75%	97%	45%	98%	98%	78%	98%	99%
130	**30 m**	10%	29%	35%	28%	42%	53%	55%	75%	78%
	50 m	13%	39%	64%	37%	78%	90%	67%	92%	94%
	80 m	13%	75%	97%	45%	98%	98%	78%	98%	99%
160	**30 m**	11%	29%	36%	31%	61%	68%	59%	79%	81%
	50 m	15%	56%	90%	38%	87%	94%	76%	96%	96%
	80 m	14%	80%	94%	44%	96%	97%	79%	99%	100%
200	**30 m**	11%	31%	49%	28%	64%	77%	57%	88%	92%
	50 m	17%	62%	89%	42%	94%	98%	73%	97%	99%
	80 m	14%	77%	99%	49%	99%	100%	77%	100%	100%

Summarizing, the standard scenario (communication range of 50 meters, beacon proportion of 25%) shows average fitness values within a range of $[36, 54]$ for all scenario populations, and within $[51, 62]$ when only taking into account nodes with $F \geq 0$. Both fitness ranges are within operating levels for the model, although only about 38% of non-beacons (on average, across all scenario populations) reached the expected fitness threshold of $F_{>60}$, while another 50% remained below that, but above $F_{>20}$. Note that these values are reasonable, considering the volatility of mobile outdoor scenarios.

In terms of energy consumption, the amounts of energy utilized by the model during this stage had no statistically significant variation from those shown in the discussion of the previous experiment. The tweaks made to the fitness and decay formulae helped improve the *fitness* and *amount of nodes in scenario* indicators, but they had almost no effect over the actual energy consumption. This is due to the model working basically in the same manner as in earlier versions in terms of communication, showing only slight improvements over results from previous stages of development. However, this improvement was not significant enough to be considered an actual “enhancement”, but rather a statistical occurrence within the expected deviation.

6.4. Revisiting the Research Questions and the Hypotheses

This subsection revisits the research questions and the hypotheses stated to address the problem of providing positioning to all or most devices in a MANET, in mobile outdoor environments, even if they are somehow impeded to do so.

We proposed that in most cases, participating devices commit to a single positioning strategy, even when there are better options. Moreover, not all of the participating devices possess peripherals that enable them to perform positioning on their own, or they could have their peripherals turned off. In addition, we stated that energy is usually a scarce asset in mobile environments, and that aspect must be taken into account during the positioning process. With these statements in mind, two research questions were postulated to drive our research forward:

RQ1: How to enable all or most of devices in a given outdoor scenario to estimate their positions, even if they have limited or no positioning capabilities of their own?

RQ2: How to achieve the former without increasing considerably the energy consumption of both individual devices and the entire network?

In order to tackle these questions and solve the aforementioned positioning problem, we proposed the following three hypotheses:

H1: Devices in outdoor environments could use their peripherals and communications capabilities to sense their environment, effectively assembling a working-context that includes all elements relevant to the positioning process.

H2: Devices with positioning capabilities could make use of their contextual information to determine which positioning strategy is better suited given their current context, and taking into account factors such as accuracy, response time and energy consumption.

H3: Devices with no self-positioning capabilities that are part of a mobile network could collaborate with their close neighbors, sharing positioning information (if available) and using it to obtain a rough estimate of their position.

Concerning the research questions, we answered *RQ1* by providing “equality of access” to all the participants of a MANET, in the sense that devices are not inherently limited to specific positioning strategies (e.g., using only GPS, or both GPS and RFID, but not other available strategies). Instead, every device is considered to have the possibility to access *any* positioning strategy available in its surrounding context from the start. Then, based on the device’s capability to interact with its environment (through its peripherals), this set of strategies is reduced only to those positioning strategies the device can actually access.

The results presented in Tables 6-8 and 6-10 for the standard scenario (i.e., the one that most closely resembles a real world outdoor environment) show that at least 37% of the devices for each population had fitness of over 60 units ($F_{>60}$). This means that 37% of the population (including beacons) has access to reliable positioning information. An additional 41% of devices have fitness lower than 60 but greater than 20 (i.e., within minimum operating levels), and 12% with fitness below 20 (i.e., excluded from the

positioning process).

Although we were unable to provide *all* the devices in the scenario with consistent access to their position, we managed to achieve a total of 78% nodes with working fitness levels, and an additional 12% of nodes that at some point obtained their positions, but have since been unable to perform a new estimation.

With this, an average total of 90% of the nodes across all scenarios had at some point access to their positions regardless of their capabilities. Therefore, devices with positioning capabilities that are in some way impeded to perform positioning could perform positioning through other means, and devices with self-positioning capabilities could attempt to use a different positioning strategy than their default.

Summarizing, it is possible to enable most devices in a given outdoor scenario to estimate their positions regardless of their positioning capabilities, which essentially provides an answer to *RQ1*.

As for the second research question (*RQ2)*, at network level, there is an unavoidable increase in energy consumption due to the additional communication attempts performed by requesting non-beacons. This is barely offset by energy saving measures (such as, the “hold” mode), but it remains as an additional cost that does not occur in scenarios populated only by self-positioning devices.

Thus, we answered *RQ2*, but it did not achieve the energy consumption goal we proposed. The energy consumption of the model is only slightly higher than that of using plain GPS, and we had proposed that it would be at most equal, and preferably lower. A more detailed explanation is provided further down this section, during the discussion of goal fulfillment.

Regarding the stated hypotheses, by answering the research questions, we validated all three of them to some extent. By utilizing the communication and navigation peripherals of the devices, the CAMPOS model provides devices with the ability to sense and characterize their working-context. With this, the model can assemble the device and scenario feature vectors (shown in Section 4.2) that are later used to determine the positioning recommendations, therefore validating the first hypothesis (*H1*).

As for hypothesis *H2*, CAMPOS provides devices with the possibility to asses all of the positioning strategies available to them, determining which is better suited based on the assembled feature vectors that represent their working context. A random forest classifier (Section 4.2) then uses these vectors to determine the more suitable positioning strategy, and once the positioning process is finished, the model provides the requesting applications with the position of the device. Factors such as the accuracy requirement of the requesting application, and the energy levels of the devices, are restrictions imposed on the model, which endorses the validation of the second hypothesis (*H2*).

Finally, related to the third hypothesis (*H3)*, the model potentially enables all devices without self-positioning capabilities to perform positioning collaboratively, as well as those with self-positioning capabilities that for some reason are unable to perform positioning. Results show that 90% of the devices in a given scenario knew their positions at some point during the simulation, while a total of 78% know their positions with fitness within operating levels. Thus, we can safely say that hypothesis *H3* has been thoroughly validated.

In terms of the thesis objective, we fulfilled the main objective by providing most of the participant devices with the capacity of performing outdoor positioning, regardless of their inherent (or inexistent) positioning capabilities. CAMPOS allows devices to sense changes in their context, and adapt their behavior accordingly, either by recommending the use of a positioning strategy, or by removing the device from the collaboration. This ability, of course, is limited by the device's sensing capabilities.

The model provides self-positioning devices with a "menu" of positioning capabilities, allowing them to pick the one that works better for their surrounding context, instead of using the default one (generally GPS). As for devices without positioning capabilities and self-positioning devices unable to access positioning strategies, the model provides them with the capacity to perform collaborative positioning based on their neighbors positioning information.

In addition, CAMPOS requires no additional instrumentation, basing its functions on strategies and reference points (i.e., neighboring devices and access points) that the requesting devices can sense in the environment. If a device is unable to sense a particular positioning strategy, it can still benefit from it via collaboration, through a surrogate device that has access to said positioning strategy.

The only goal that was not entirely fulfilled (with respect to our initial expectations) is that of avoiding unnecessary energy consumption. Although the model allows devices not to attempt to perform positioning if conditions are not suitable, the amount of communications required to perform collaborative positioning, as well as the energy required to perform context-sensing, tend to increase the energy consumption levels of all participant devices.

Results show that an additional 184 to 306 energy units are spent by devices utilizing the model actively for a lapse of 5 minutes, with respect to devices that do not (Figure 6-8). These amounts are not high in terms of battery energy, considering that the model works in short bursts of activity, and then it "sleeps" waiting for fitness to decay before attempting a new positioning request. However, continuous use of the model over prolonged time lapses is sure to bring down the battery of a device. This latent issue should be tackled as part of the future work.

Chapter 7: Conclusions and Future Work

This thesis presents CAMPOS, a context-aware positioning model for outdoor scenarios. The model provides mobile devices with the means to determine their position based on contextual elements such as neighboring devices, access points, and available positioning strategies, to name a few. The model was tested using simulated outdoor environments, consisting of 500 × 500 m^2 areas populated by three types of nodes: pedestrians, vehicles, and stationary. These nodes have roles depending on their ability to use positioning strategies; beacons have access to at least one positioning strategy, and non-beacons have no access whatsoever.

CAMPOS enables up to 90% of devices in an outdoor scenario to estimate their position based on information gathered from their surrounding context, specifically the population of reference points (i.e., access points and other devices with known positions) and available positioning strategies. In addition, devices with access to more than one positioning strategy are provided with the ability to choose the strategy that works better than the default one (mainly GPS) for their particular context.

The model succeeds in providing a degree of positioning to mobile applications running on devices otherwise unable to position themselves. This goal is reached by sharing positioning information with the neighboring nodes. However, due to the nature of the collaboration (i.e., the reference points' positions have a variable error), the accuracy of the estimations is expected to be significantly lower than using a positioning strategy directly. Based on literature, the expected collaboration errors could range from less than 1 meter to over 12 meters above the error associated to GPS. Note that this positioning error applies to devices that would be unable to position themselves if not for CAMPOS.

Likewise, by using the set of recommendations assembled by the model, a device is able to access any of the positioning strategies available in its surroundings. This eliminates the need to start the whole positioning process over when there has been no significant change in context, since the device can use any of the recommended strategies. The interval at which the model senses the environment depends on the speed of the device, and on the position accuracy requirement of the consumer application.

In terms of energy, despite advances in battery technology, maintaining an active connection (e.g., GPS or Wi-Fi) consumes significant amounts of power. Thus, the model must make additional decisions to ensure extended operation to the devices, e.g., sensing the context of a device to determine whether it is moving or not, in order to avoid requesting an unnecessary positioning estimation, or determining whether a device has enough energy to go through the positioning process without jeopardizing the device's basic functions.

Given that most proposals are focused on providing novel approaches to increase the accuracy of the positioning estimates, there is little room for comparison with other

proposals in terms of how many nodes benefit from the use of our model. In addition, since the fitness metric is a novel proposal that represents a state of “validity” of a positioning estimation, and not its accuracy, it is not possible to compare on those grounds, either.

The proposal of Savarese et al. [143] is one of the few collaborative proposals that provides actual numbers for amounts of nodes that achieve positioning. About 45% of the population with their positions, as long as at least five reference points are available for each target node. This number increases closer to 95% when there are 12 reference points. Note that their proposal requires the use of a specific positioning algorithm, Hop-TERRAIN with (and without) refinement, and that their scenario consists on a 200 × 200 meter scenario populated by 400 nodes, a scenario smaller than ours and with a larger population. However, their proposal requires only 5% of beacons present in a scenario.

In addition, although we did not achieve our goal of keeping the energy consumption at the same level as that of GPS (or lower), the additional energy expended is not considerable. However, given the nature of mobile environments, it is not negligible, and must be tackled in future iterations of the model.

The future work involves conducting an evaluation of the energy consumption of the model, seeking to improve the consumption levels not only to node level, but also considering the whole interconnected system. Several settings of the application scenario should be simulated and analyzed. The obtained results should help us improve the positioning priorities considered during the strategy choosing process performed by the model. Security and privacy issues related to the shared information are also some of the challenges to address during the next step. In addition, an optimal context-sensing interval must be found in order to ensure less energy usage.

Another aspect that should be tackled in future work is a test of the performance of CAMPOS in a real world environment. Although the ns-3, the simulator chosen as our experimental test-bed, has been tested extensively with great success by the research community, a real world experiment is still necessary in order to corroborate the validity of our context-aware positioning model.

References

[1] Abowd, G. D.; Dey, A. K.; Brown, P. J.; Davies, N.; Smith, M.; Steggles, P. (**1999**). *Towards a better understanding of context and context-awareness*. In Handheld and ubiquitous computing, pp. 304–307, Springer Berlin Heidelberg.

[2] Ahas, R.; Aasa, A.; Roose, A.; Mark, Ü.; Silm, S. (**2008**). *Evaluating passive mobile positioning data for tourism surveys: An Estonian case study*. Tourism Management, 29(3), pp. 469–486.

[3] Ahas, R.; Silm, S.; Järv, O.; Saluveer, E.; Tiru, M. (**2010**). *Using mobile positioning data to model locations meaningful to users of mobile phones*. Journal of Urban Technology, 17(1), pp. 3–27.

[4] Aschenbruck, N.; Gerhards-Padilla, E.; Martini, P. (**2009**). *Modeling mobility in disaster area scenarios*. Performance Evaluation, 66(12), pp. 773–790.

[5] Aschenbruck, N.; Ernst, R.; Gerhards-Padilla, G.; Schwamborn, M. (**2010**). *BonnMotion: A mobility scenario generation and analysis tool*. In Proceedings of the 3rd Int. ICST Conference on Simulation Tools and Techniques, Malaga, Spain, pp. 15–19.

[6] Baig, R.; Roca, R.; Freitag, F., Navarro, L. (**2015**). *Guifi.net, a crowdsourced network infrastructure held in common*. Computer Networks, 90, pp. 150–165.

[7] Bao, J.; Zheng, Y.; Mokbel, M.F. (**2012**). *Location-based and preference-aware recommendation using sparse geo-social networking data*. In Proceedings of the 20th ACM Int. Conference on Advances in Geographic Information Systems, pp. 199–208.

[8] Bardram, J.E. (**2004**). *Applications of context-aware computing in hospital work: examples and design principles*. In Proceedings of the 2004 ACM symposium on Applied computing, pp. 1574–1579.

[9] Barequet, G.; Har-Peled, S. (**2001**). *Efficiently approximating the minimum-volume bounding box of a point set in three dimensions*. Journal of Algorithms, 38(1), pp. 91–109.

[10] Beauregard, S.; Haas, H. (**2006**). *Pedestrian dead reckoning: A basis for personal positioning*. In Proceedings of the 3rd Workshop on Positioning, Navigation and Communication, Hannover, Germany, pp. 22-22.

[11] Bellavista, P.; Corradi, A.; Fanelli, M.; Foschini, L. (**2012**) *A Survey of Context Data Distribution for Mobile Ubiquitous Systems*. In ACM Computing Surveys, 44(4), pp. 24.

[12] Benítez-Guerrero, E.; Mezura-Godoy, C.; Montané-Jiménez, L.G. (**2012**). *Context-aware mobile collaborative systems: Conceptual modeling and case study*. Sensors Journal, 12(10), pp. 13491–13507.

[13] Bianchi, G. (**2000**). *Performance analysis of the IEEE 802.11 distributed coordination*

function. IEEE Journal on selected areas in communications, 18(3), pp. 535–547.

[14] Böse, F.; Piotrowski, J.; Scholz-Reiter, B. (**2009**). *Autonomously controlled storage management in vehicle logistics–applications of RFID and mobile computing systems*. International Journal of RF Technologies: Research and Applications, 1(1), pp. 57–76.

[15] Bradler, D.; Schiller, B. (**2009**). *Towards a distributed crisis response communication system*. In Proceedings of the ISCRAM, Göteborg, Sweden, pp. 10–13.

[16] Braunstein, B.; Trimble, T.; Mishra, R.; Manoj, B.S.; Lenert, L.; Rao, R. (**2006**). *Challenges in using distributed wireless mesh networks in emergency response*. In Proceedings of the ISCRAM, Newark, NJ, USA, pp. 13–17.

[17] Bravo, J.; Hervás, R.; Sánchez, I.; Chavira, G.; Nava, S. (**2006**). *Visualization services in a conference context: an approach by RFID technology*. Journal of Universal Computer Science, 12(3), pp. 270-283.

[18] Breiman, L.; Cutler, A. *Random Forests*. Salford Systems. Available at: www.stat.berkeley.edu/~breiman/RandomForests/ [Visited: 2016-12]

[19] Brézillon, P. (**2003**). *Individual and team contexts in a design process*. In Proceedings of the 36th IEEE Hawaii International Conference on System Sciences, HICSS-36, Track "Collaboration Systems and Technology", R.H. Sprague, Los Alamitos.

[20] Brézillon, P.; Borges, M.; Pino, J.A.; Pomerol, J.C. (**2004**). *Context-awareness in group work: Three case studies*. In Proceedings of IFIP International Conference on Decision Support Systems, Prato, Italy, pp. 1–3.

[21] Brumitt, B.; Meyers, B.; Krumm, J.; Kern, A.; Shafer, S. (**2000**). *Easyliving: Technologies for intelligent environments*. Handheld and Ubiquitous Comp., Springer Press, pp. 97–119.

[22] Bulusu, N.; Heidemann, J.; Estrin, D. (**2000**). *GPS-less low-cost outdoor localization for very small devices*. IEEE Personal Communications, 7(5), pp. 28–34.

[23] Burgard, W.; Derr, A.; Fox, D.; Cremers, A.B. (**1998**). *Integrating global position estimation and position tracking for mobile robots: the dynamic Markov localization approach*. In Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2, pp.730–735.

[24] Burke, J.; Estrin, D.; Hansen, M.; Parker, A.; Ramanathan, N.; Reddy, S.; Srivastava, M.B. (**2006**). *Participatory sensing*. In Workshop on World-Sensor-Web (WSW), Mobile Device Centric Sensor Networks and Applications.

[25] Camp, T.; Boleng, J.; Davies, V. (**2002**). *A survey of mobility models for ad hoc network research*. Wireless Communications and Mobile Computing, 2, pp. 483–502.

[26] Čapkun, S.; Hamdi, M.; Hubaux, J.P. (**2002**). *GPS-free positioning in mobile ad hoc*

networks. Cluster Computing, 5, pp. 157–167.

[27] Carreño, P.; Gutierrez, F.J.; Ochoa, S.F.; Fortino, G. (**2015**). *Supporting personal security using participatory sensing*. Concurrency and Computation: Practice and Experience, 27(10), pp. 2531–2546.

[28] Carroll, A.; Heiser, G. (**2010**). *An analysis of power consumption in a smartphone*. In Proceedings of USENIX 2010, Boston, MA, USA, 22–25.

[29] CCS Insight: Wearables Momentum Continues (**2016**). Available at: http://www.ccsinsight.com/press/company-news/2516-wearables-momentum-continues [Visited: 2016-12].

[30] Chen, J.C.; Wang, Y.C.; Maa, C.S.; Chen, J.T. (**2006**). *Network-side mobile position location using factor graphs*. IEEE Transactions on Wireless Communications, 5(10), pp. 2696–2704.

[31] Chen, Z.; Dong, W.; Li, H.; Zhang, P.; Chen, X.; Cao, J. (**2014**). *Collaborative network security in multi-tenant data center for cloud computing*. Tsinghua Science and Technology, 19(1), 82–94.

[32] Cheverst, K.; Davies, N.; Mitchell, K.; Friday, A.; Efstratiou, C. (**2000**). *Developing a context-aware electronic tourist guide: some issues and experiences.* In Proceedings of the ACM SIGCHI conference on Human Factors in Computing Systems, pp. 17–24.

[33] Chon, H.D.; Jun, S.; Jung, H.; An, S.W. (**2004**). *Using RFID for accurate positioning*. Journal of Global Positioning Systems, 3(1–2), pp. 32–39.

[34] Chung, K.Y.; Yoo, J.; Kim, K.J. (**2014**). *Recent trends on mobile computing and future networks*. Personal and Ubiquitous Computing 18(3), pp. 489–491.

[35] Churchill, E.; Wakeford, N. (**2001**). *Framing mobile collaborations and mobile technologies.* In Wireless world: Social and Interactional Aspects of the Mobile Age, Springer-Verlag, New York, pp. 154–179.

[36] Cook, B.; Buckberry, G.; Scowcroft, I.; Mitchell, J.; Allen, T. (**2005**). *Indoor location using trilateration characteristics*. In Proceedings of the London Communications Symposium, pp. 147–150.

[37] Cristianini, N.; Shawe-Taylor, J. (**2000**). *An introduction to support Vector Machines and other kernel-based learning methods*. Cambridge University Press.

[38] CSUAC Art collection. Available at: http://www.vintyri.org/CSUAC/ [Visited: 2016-12]

[39] De-Maesschalck, R.; Jouan-Rimbaud, D.; Massart, D.L. (**2000**). *The mahalanobis distance*. Chemometrics and Intelligent Laboratory Systems, 50(1), pp. 1–18.

[40] Devin Night, Custom art and Tokens for digital tabletop roleplaying, Available at:

https://immortalnights.com/tokensite/ [Visited: 2016-12]

[41] Dey, A.K. (**1998**) *Context-Aware Computing: The CyberDesk Project*. AAAI Spring Symposium on Intelligent Environments, Technical Report SS-98-02, pp. 51–54.

[42] Dickinson, J.E.; Ghali, K.; Cherrett, T.; Speed, C.; Davies, N.; Norgate, S. (**2014**). *Tourism and the smartphone app: Capabilities, emerging practice and scope in the travel domain*. Current Issues in Tourism, 17(1), pp. 84–101.

[43] Dourandish, B.; Zumel, N.; Manno, M. (**2007**) *Command and control during the first 72 hours of a joint military-civilian disaster response*. In proceedings of the Command and Control Research and Technology Symposium, Newport, RI, USA, pp. 19–21.

[44] Dourish, P.; Bellotti, V. (**1992**) *Awareness and Coordination in Shared Workspaces*. In Proceedings of the ACM Conference on Computer-Supported Cooperative Work CSCW'92, Toronto, Ontario, pp.107–114.

[45] Doshi, P.; Jain, P.; Shakwala, A. (**2014**). *Location based services and integration of google maps in android*. Int. J Eng Computer Sci, 3(3), pp. 5072–5077.

[46] Eltahir, I.K. (**2007**) *The impact of different radio propagation models for mobile ad hoc networks (manet) in urban area environment*. In Proceedings of the 2nd International Conference on Wireless Broadband and Ultra-Wideband Communications, Sydney, Australia, pp. 27–30.

[47] Eng, T.Y. (**2006**). *Mobile supply chain management: Challenges for implementation*. Technovation, 26(5), pp. 682–686.

[48] Evennou, F.; Marx, F. (**2006**). *Advanced integration of WiFi and inertial navigation systems for indoor mobile positioning*. Eurasip Journal on Applied Signal Processing, pp. 164-164.

[49] Fang, B.T. (**1990**). *Simple solutions for hyperbolic and related position fixes*. IEEE Transactions on Aerospace and Electronic Systems, 26(5), pp. 748–753.

[50] Federal Communications Commission – FCC (**2011**). *The Role of Deployable Aerial Communications Architecture in Emergency Communications and Recommended Next Steps*. Public Safety and Homeland Security Bureau, Washington, DC, USA.

[51] Ficco, M.; Palmieri, F.; Castiglione, A. (**2014**). *Hybrid indoor and outdoor location services for new generation mobile terminals*. Personal and Ubiquitous Computing, 18(2), pp. 271-285.

[52] Fickas, S.; Korteum, G.; Segall, Z. (**1997**) *Software Organization for Dynamic and Adaptable Wearable Systems*. 1st Int. Symposium on Wearable Computers, pp. 56–63.

[53] Fitrianie, S.; Rothkrantz, L. (**2015**). *Dynamic Routing during Disaster Events*. In Proceedings of the ISCRAM, Kristiansand, Norway, pp. 24–27.

[54] Fitzgerald, E.; Nyberg, C.; Priyanto, B.E. (**2016**). *Modeling, implementation and evaluation of IEEE 802.11 ac in NS-3 for enterprise networks*. In 2016 IEEE Wireless Days (WD), pp. 1–6.

[55] Ganti, R.K.; Ye, F.; Lei, H. (**2011**). *Mobile crowdsensing: current state and future challenges*. IEEE Communications Magazine, 49(11), pp. 32–39.

[56] Giaglis, G. M.; Minis, I.; Tatarakis, A.; Zeimpekis, V. (**2004**). *Minimizing logistics risk through real-time vehicle routing and mobile technologies: Research to date and future trends*. Int. Journal of Physical Distribution & Logistics Management, 34(9), pp. 749–764.

[57] Giordano, S. (**2002**). *Mobile ad hoc networks*. Handbook of wireless networks and mobile computing, pp. 325-346.

[58] Gregoski, M. J.; Mueller, M.; Vertegel, A.; Shaporev, A.; Jackson, B.B.; Frenzel, R.M.; ...; Treiber, F.A. (**2012**). *Development and validation of a smartphone heart rate acquisition application for health promotion and wellness telehealth applications*. International journal of telemedicine and applications, 1.

[59] Gu, Y.; Lo, A.; Niemegeers, I. (**2009**). *A survey of indoor positioning systems for wireless personal networks*. IEEE Communications Surveys & Tutorials, 11(1), pp. 13–32.

[60] Guerrero, L.A.; Ochoa, S.F.; Pino, J.A.; Collazos, C.A. (**2006**). *Selecting computing devices to support mobile collaboration*. Group Decision and Negotiation, 15(3), 243–271.

[61] Günther, A.; Hoene, C. (**2005**). *Measuring round trip times to determine the distance between WLAN nodes*. In Proceedings of the 4th IFIP-TC6 international conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communication Systems, Waterloo, Canada, pp. 768–779.

[62] Guo, B.; Wang, Z.; Yu, Z.; Wang, Y.; Yen, N.Y.; Huang, R.; Zhou, X. (**2015**). *Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm*. ACM Computing Surveys (CSUR), 48(1), pp. 7.

[63] Gutwin, C.; Greenberg, S. (**2002**). *A descriptive framework of workspace awareness for real-time groupware*. Computer Supported Cooperative Work (CSCW), 11(3–4), pp. 411–446.

[64] Hall, D.L.; Llinas, J. (**1997**). *Introduction to multisensor data fusion*. In Proceedings of the IEEE, 85, pp. 6–23.

[65] Haykin, S. (**2008**). *Communication systems (4° Ed.)*. John Wiley & Sons. pp. 488–99. Retrieved 11 April 2015.

[66] Heathers, J.A. (**2013**). *Smartphone-enabled pulse rate variability: an alternative methodology for the collection of heart rate variability in psychophysiological research*. International Journal of Psychophysiology, 89(3), pp. 297–304.

[67] Henderson, T.R.; Roy, S.; Floyd, S.; Riley, G.F. (**2006**). *ns-3 Project Goals*. In Proceedings of the ACM 2006 workshop on ns-2: the IP network simulator, Pisa, Italy.

[68] Herskovic, V.; Ochoa, S.F.; Pino, J.A.; Neyem, A. (**2011**). *The Iceberg Effect: Behind the User Interface of Mobile Collaborative Systems*. Journal of Universal Computer Science 17(2), pp. 183-202.

[69] Hightower, J.; Borriello, G. (**2001**). *Location systems for ubiquitous computing*. IEEE Computer, 34(8), pp. 57–66.

[70] Hofmann-Wellenhof, B.; Lichtenegger, H.; Collins, J. (**1993**). *Global Positioning System: Theory and Practice*. Springer Press, Wien, Austria.

[71] Hofmann-Wellenhof, B.; Lichtenegger, H.; Wasle, E. (**2008**). *GNSS—Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and More*. Springer Science & Business Media, Vienna, Austria.

[72] Hong, X.; Gerla, M.; Pei, G.; Chiang, C.C. (**1999**). *A group mobility model for ad hoc wireless networks*. In Proceedings of the 2nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Seattle, WA, USA, pp. 20–20.

[73] Huang, B.; Yao, Z.; Cui, X.; Lu, M. (**2016**). *Dilution of Precision Analysis for GNSS Collaborative Positioning*. IEEE Transactions on Vehicular Tech., 65(5), pp. 3401-3415.

[74] Hull, R.; Neaves, P.; Bedford-Roberts, J. (**1997**). *Towards Situated Computing*. 1st International Symposium on Wearable Computers, pp. 146–153.

[75] IDC Analyze the Future (**2016**). *IDC Forecasts Worldwide Shipments of Wearables to Surpass 200 Million in 2019, Driven by Strong Smartwatch Growth and the Emergence of Smarter Watches.* Available at: http://www.idc.com/getdoc.jsp?containerId=prUS41530816 [Visited: 2016-12].

[76] Jacquet, P.; Muhlethaler, P.; Clausen, T.; Laouiti, A.; Qayyum, A.; Viennot, L. (**2001**). *Optimized link state routing protocol for ad hoc networks*. In Proceedings of the IEEE International Multi-Topic Conference: Technology for the 21st Century (INMIC 2001), Lahore, Pakistan, pp. 62–68.

[77] Järvinen, R.; Jaakkola, A.; Määttä, J.; Liuhto, L.; Luostarinen, R.; Manner, J.; Luoma, M. (**2015**). *Hierarchical link-state routing in disruption-tolerant networks*. In 2nd IEEE World Symposium on Web Applications and Networking (WSWAN), pp. 1–7.

[78] Ji, X.; Zha, H. (**2004**). *Sensor positioning in wireless ad-hoc sensor networks using multidimensional scaling*. INFOCOM 2004: 23th Annual Joint Conference of the IEEE Computer and Communications Societies, 4, pp. 2652–2661.

[79] Jiang, X.; Chen, N.Y.; Hong, J.I.; Wang, K.; Takayama, L.; Landay, J.A. (**2004**). *Siren: Context-aware computing for firefighting*. In International Conference on Pervasive

Computing, Springer Berlin Heidelberg, pp. 87–105.

[80] Jing, H.; Pinchin, J.; Hill, C.; Moore, T. (**2016**). *An adaptive weighting based on modified DOP for collaborative indoor positioning*. J. of Navigation, 69(2), pp. 225–245.

[81] Kaemarungsi, K.; Krishnamurthy, P. (**2004**). *Properties of indoor received signal strength for WLAN location fingerprinting*. MOBIQUITOUS 2004: The First Annual Int. Conference on Mobile and Ubiquitous Systems: Networking and Services, pp. 14–23.

[82] Kanaan, M.; Pahlavan, K. (**2004**). *A comparison of wireless geolocation algorithms in the indoor environment*. IEEE Wireless Communications and Networking Conference WCNC 2004, 1, pp. 177–182.

[83] Kealy, A.; Retscher, G.; Toth, C.; Brzezinska, D. (**2014**). *Collaborative Positioning–Concepts and Approaches for More Robust Positioning*. In Proceedings of the XXV International FIG Congress, pp. 16–21.

[84] Kim, D.; Kim, J.H.; Moon, C.; Choi, J.; Yeom, I. (**2016**). *Efficient content delivery in mobile ad-hoc networks using ccn*. Ad Hoc Networks, 36, pp. 81–99.

[85] Kontkanen, P.; Myllymaki, P.; Roos, T.; Tirri, H.; Valtonen, K.; Wettig, H. (**2004**). *Topics in probabilistic location estimation in wireless networks*. 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications PIMRC 2004, 2, pp. 1052–1056.

[86] Koukoumidis, E.; Martonosi, M.; Peh, L.S. (**2012**). *Leveraging smartphone cameras for collaborative road advisories*. IEEE Transactions on Mobile Comp., 11(5), pp. 707–723.

[87] Ladd, A.M.; Bekris, K.E.; Rudys, A.P.; Wallach, D.S.; Kavraki, L.E. (**2004**). *On the feasibility of using wireless ethernet for indoor localization*. IEEE Transactions on Robotics and Automation, 20(3), pp. 555–559.

[88] Lassabe, F.; Canalda, P.; Chatonnay, P.; Spies, F.; Baala, O. (**2005**). *A Friis-based calibrated model for WiFi terminals positioning*. In Proceedings of the 6th IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks, Taormina, Giardini Naxos, Italy.

[89] Lee, K.; Hong, S.; Kim, S.J.; Rhee, I.; Chong, S. (**2010**). *Slaw: A new mobility model for human walks*. In Proceedings of INFOCOM 2009, Rio de Janeiro, Brazil.

[90] Lee, E.B. (**2015**). *Too Much Information Heavy Smartphone and Facebook Utilization by African American Young Adults*. Journal of Black Studies, 46(1), 44–61.

[91] Leung, S.Y.; Montenbruck, O.; Bruninga, B. (**2001**). Hot start of GPS receivers for LEO microsatellites.

[92] Liang, C.; Yu, F.R. (**2015**). *Wireless network virtualization: A survey, some research issues and challenges*. IEEE Communications Surveys & Tutorials, 17(1), pp. 358–380.

[93] Lilien, L.; Kamal, Z. H.; Gupta, A.; Bhuse, V.; Yang, Z. (**2006**). *Opportunistic sensor networks (oppnets)*. In Proceedings of the 3rd International Conference on Networked Sensing Systems (INSS'06).

[94] Liu, H.; Darabi, H.; Banerjee, P.; Liu, J. (**2007**). *Survey of Wireless Indoor Positioning Techniques and Systems*. IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews, 37(6), pp. 1067–1080.

[95] Logistix Partners Oy (**1996**), *Bussiness Logistics, Defiinitions of Logistics*. Helsinki, FI.

[96] Loh, S.; Lorenzi, F.; Saldaña, R.; Licthnow, D. (**2003**). *A tourism recommender system based on collaboration and text analysis*. Information Tech. & Tourism, 6(3), pp. 157–165.

[97] López-de-Ipiña, D.; Ochoa, S.F.; Bravo, J. (**2013**). *Preface to the special section on Software Engineering Aspects of Ubiquitous Computing and Ambient Intelligence (UCAmI 2011)*. Science of Computer Programming, 10(78), pp. 1892–1894.

[98] Lu, F.; Milios, E. (**1997**). *Globally consistent range scan alignment for environment mapping*. Journal on Autonomous Robots, Springer Press, 4(4), pp. 333–349.

[99] Manoj, B.S.; Baker, A. (**2007**). *Communication challenges in emergency response*. Communications of the ACM, 45, pp. 51–53.

[100] Mansfield, K.C.; Antonakos, J.L. (**2010**). *Computer Networking from LANs to WANs: Hardware, Software, and Security*. Cengage Learning.

[101] Marine Navigation app. Available at: https://itunes.apple.com/us/app/marine-navigation-netherlands/id618506531?mt=8 [Visited: 2016-12]

[102] Mary Meeker. Internet Trends 2015 – Code Conference Report. KPCB. Available at: http://www.kpcb.com/internet-trends [Visited: 2016-12]

[103] Maybeck, P.S. (**1990**). *The Kalman filter: An introduction to concepts*. Autonomous Robot Vehicles, Springer Press Heidelberg, pp. 194–204.

[104] Medeisis, A.; Kajackas, A. (**2000**). *On the use of the universal Okumura-Hata propagation prediction model in rural areas*. In Proceedings of the IEEE 51th Vehicular Technology Conference, Tokyo, Japan, pp. 15–18.

[105] Meguro, J.I.; Hashizume, T.; Takiguchi, J.I.; Kurosaki, R. (**2005**). *Development of an autonomous mobile surveillance system using a network-based RTK-GPS*. In Proceedings of the 2005 IEEE Int. Conf. on Robotics and Automation, pp. 3096–3101.

[106] Mendonça, D. (**2007**). *Decision support for improvisation in response to extreme events: learning from the response to the 2001 world trade center attack*. Decision Support Systems, 43, pp. 952–967.

[107] Messeguer, R.; Navarro, L.; Damian-Reyes, P.; Favela, J. (**2010**). *Context Awareness for*

Collaborative Learning with Uncertainty Management. J. UCS, 16(12), pp. 1556–1576.

[108] Miluzzo, E. (**2011**) *Smartphone sensing*. PhD thesis. Dartmounth College Hanover, New Hampshire.

[109] Montemerlo, M.; Thrun, S.; Koller, D.; Wegbreit, B. (**2002**). *FastSLAM: a factored solution to the simultaneous localization and mapping problem*. 18th National Conference on Artificial intelligence, pp. 593–598.

[110] Montemerlo, M.; Thrun, S.; Koller, D.; Wegbreit, B. (**2003**). *FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges*. In Proceedings of the 18th International Joint conference on Artificial intelligence IJCAI'03, Morgan Kaufmann Publishers, pp. 1151–1156.

[111] Moreno, D.; Ochoa, S.F.; Santos, R.; Meseguer, R. (**2013**). *Geo-localized messages irradiation using smartphones: An energy consumption analysis*. IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2013), Whistler, BC, Canada, pp. 681-685.

[112] Moreno, D.; Ochoa, S.F.; Meseguer, R. (**2014**). *Understanding confidence of positioning measurements in collaborative outdoor environments*. IEEE International Conference on Systems, Man and Cybernetics (SMC2014), San Diego, CA, USA, pp. 1141-1146.

[113] Moreno, D.; Ochoa, S.F.; Meseguer, R. (**2015**). *Providing ubiquitous positioning in outdoor environments.* IEEE International Conference on Systems, Man and Cybernetics (SMC2015), Hong Kong, China, pp. 1285-1290.

[114] Moreno, D.; Ochoa, S.F.; Meseguer, R. (**2015**). *A context-aware model to provide positioning in disaster relief scenarios*. Sensors 15(10), pp. 25176–25207.

[115] Moreno, D.; Ochoa, S.F. (**2016**). *Understanding the Resource Positioning Methods that Support Mobile Collaboration*. IEEE International Conference on Systems, Man and Cybernetics (SMC2016), Budapest, Hungary, pp. 3676-3682.

[116] Morgan Stanley Research (**2009**). *Morgan Stanley The Mobile Internet Report*. Morgan Stanley & Co.

[117] Nam, S.Z. (**2013**). *Evaluation of University Students' Utilization of Smartphone*. International Journal of Smart Home, 7(4), pp. 162–173.

[118] Ning, C.; Li, R.; Li, K. (**2016**). *Outdoor Location Estimation Using Received Signal Strength-Based Fingerprinting*. Wireless Personal Communications 89(2), pp. 1–20.

[119] Ochoa, S.F.; Neyem, A.; Pino, J.A.; Borges, M. (**2007**). *Supporting group decision making and coordination in urban disasters relief efforts*. Journal of Decision Systems, 16, pp. 143–172.

[120] Ochoa, S.F.; Pino, J.A. (**2008**). *Challenges for Decision Support in Urban Disaster Scenarios*. Encyclopedia of Decision Making and Decision Support Technologies, Hershey, PA, USA.

[121] Ochoa, S.F.; Santos, R. (**2015**). *Human-centric wireless sensor networks to improve information availability during urban search and rescue activities*. In Information Fusion, 22, pp. 71–84.

[122] Oliver, M.; Badland, H.M.; Mavoa, S.; Duncan, M.J.; Duncan, J.S. (**2010**). *Combining GPS, GIS, and accelerometry: methodological issues in the assessment of location and intensity of travel behaviors*. Journal of Physical Activity and Health, 7, pp. 102–108.

[123] Omar, B.; Ballal, T. (**2009**). *Intelligent wireless web services: context-aware computing in construction-logistics supply chain*. ITcon, 14, pp. 289–308.

[124] OpenCV 2.4.13.1 documentation, API Reference, Machine Learning, Decision Trees. Available at: http://docs.opencv.org/2.4/modules/ml/doc/decision_trees.html# [Visited: 2016-12].

[125] Oulasvirta, A.; Raento, M.; Tiitta, S. (**2005**). *ContextContacts: re-designing SmartPhone's contact book to support mobile awareness and collaboration*. In Proceedings of the 7th ACM International Conference on Human Computer Interaction with mobile Devices & Services, pp. 167–174

[126] Oulasvirta, A.; Rattenbury, T.; Ma, L.; Raita, E. (**2012**). *Habits make smartphone use more pervasive*. Personal and Ubiquitous Computing, 16(1), pp. 105–114.

[127] Pang, H.; Jiang, L.; Yang, L.; Yue, K. (**2010**). *Research of android smart phone surveillance system*. In IEEE 2010 International Conference on Computer Design and Applications (ICCDA), 2, pp. 373–376.

[128] Peters, O.; Ben Allouch, S. (**2005**). *Always connected: a longitudinal field study of mobile communication*. Telematics and Informatics, 22(3), pp. 239–256.

[129] Pinelle, D.; Gutwin, C. (**2005**). *A Groupware Design Framework for Loosely Coupled Workgroups*. In Proceedings of ECSCW'05: European Conference on Computer-Supported Cooperative Work, Springer-Verlag, New York, pp. 119-139.

[130] Povalac, A.; Šebesta, J. (**2010**). *Phase of arrival ranging method for UHF RFID tags using instantaneous frequency measurement*. In Proc. of the IEEE ICECom 2010, pp. 1–4.

[131] Prasithsangaree, P.; Krishnamurthy, P.; Chrysanthis, P. (**2002**). *On indoor position location with wireless LANs*. In 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, 2, pp. 720–724.

[132] Purohith, D.R.; Hegde, A.; Sivalingam, K.M. (**2015**). *Network architecture supporting seamless flow mobility between LTE and WiFi networks*. In IEEE 16th Int. Symposium on a

World of Wireless, Mobile and Multimedia Networks (WoWMoM), pp. 1–9.

[133] Rachuri, K.K. (**2012**). *Smartphones based Social Sensing: Adaptive Sampling, Sensing and Computation Offloading*. University of Cambridge Computer Lab oratory, St. John's College.

[134] Rodden, T.; Cheverst, K.; Davies, K.; Dix, A. (**1998**). *Exploiting context in HCI design for mobile systems*. Workshop on HCI with mobile devices, pp. 21–22.

[135] Rodríguez-Covili, J.F.; Ochoa, S.F.; Pino, J.A.; Messeguer, R.; Medina, E.; Royo, D. A (**2001**). *Communication infrastructure to ease the development of mobile collaborative applications*. Journal of Network and Computer Applications, 34, pp. 1883–1893.

[136] Rodríguez-Covili, J.F.; Ochoa, S.F.; Pino, J.A. (**2012**). *High Level MANET Protocol: Enhancing the Communication Support for Mobile Collaborative Work*. Journal of Network Computer Applications, 35, pp. 145–155.

[137] Ruiz-López, T.; Garrido, J.; Benghazi, K.; Chung, L. (**2010**). *A survey on indoor positioning systems: foreseeing a quality design*. Distributed Computing and Artificial Intelligence, pp. 373–380.

[138] Runge, T.M.; Beifuß, A.; Wolfinger, B.E. (**2015**). *Low latency network traffic processing with commodity hardware*. In Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems, pp. 1–8.

[139] Ryan, N.; Pascoe, J.; Morse, D. (**1997**). *Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant*. Computer Applications in Archaeology.

[140] Ryan, N. (**1997**). *Mobile Computing in a Fieldwork Environment: Metadata Elements*. Project working document, version 0.2.

[141] Sahoo, P.K.; Hwang, I.S. (**2011**). *Collaborative localization algorithms for wireless sensor networks with reduced localization error*. Sensors, 11, pp. 9989–10009.

[142] Salber, D.; Dey, A.K.; Abowd, G.D. (**1998**). *Ubiquitous Computing: Defining an HCI Research Agenda for an Emerging Interaction Paradigm*. Georgia Tech GVU Technical Report GIT-GVU-98-01.

[143] Savarese, C.; Rabaey, J.M.; Langendoen, K. (**2002**). *Robust Positioning algorithms for distributed ad-hoc wireless sensor networks*. In Proceedings of the General Track of the Annual Technical Conference on USENIX 2002, Monterey, CA, USA.

[144] Scherp, A.; Boll, S. (**2004**). *mobileMM4U-framework support for dynamic personalized multimedia content on mobile systems*. In Proceedings des Techniques and Applications for Mobile Commerce (TaMoCO) Track der Multi-Konferenz Wirtschaftsinformatik 2004, Essen, Deutschland, 3, pp. 204–215.

[145] Schilit, B.; Theimer, M. (**1994**). *Disseminating Active Map Information to Mobile Hosts*. IEEE

Network, 8(5), pp. 22–32

[146] Schmidt-Belz, B.; Nick, A.; Poslad, S.; Zipf, A. (**2002**). *Personalized and location-based mobile tourism services*. Workshop on Mobile Tourism Support Systems in conjunction with Mobile HCI.

[147] Schmidt-Belz, B.; Laamanen, H.; Poslad, S.; Zipf, A. (**2003**). *Location-based mobile tourist services: first user experiences*. ENTER 2003: 10th International Conference on Information Technology in Travel & Tourism.

[148] Schöning, J.; Rohs, M.; Krüger, A.; Stasch, C. (**2009**). *Improving the communication of spatial information in crisis response by combining paper maps and mobile devices*. In Mobile Response, Springer Berlin Heidelberg.

[149] Schwinger, W.; Grün, C.; Pröll, B.; Retschitzegger, W.; Schauerhuber, A. (**2005**). *Context-awareness in mobile tourism guides–A comprehensive survey*. Rapport Technique. Johannes Kepler University Linz.

[150] Sedjelmaci, H.; Senouci, S.M. (**2014**). *A new Intrusion Detection Framework for Vehicular Networks*. In 2014 IEEE Int. Conf. on Communications (ICC), pp. 538–543.

[151] Shaw, B.; Shea, J.; Sinha, S.; Hogue, A. (**2013**). *Learning to rank for spatiotemporal search*. In Proceedings of the 6th ACM International Conference on Web Search and Data Mining, Rome, Italy.

[152] Shin, B.J.; Lee, K.W.; Choi, S.H.; Kim, J.Y.; Lee, W.J.; Kim, H.S. (**2010**). *Indoor WiFi Positioning System for Android-based Smartphone*. In Proceedings of the international conference on information and communication technology convergence 2010, Jeju, Korea.

[153] Skov, M. B.; Høegh, R.T. (**2006**). *Supporting information access in a hospital ward by a context-aware mobile electronic patient record*. Personal and Ubiquitous Computing, 10(4), pp. 205–214.

[154] Stoffers, M.; Riley, G. (**2012**). *Comparing the ns-3 propagation models*. In Proceedings of the IEEE 20th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), San Francisco, CA, USA.

[155] Teixeira, T.; Zink, M. (**2016**). *Evaluating Information-Centric Networks in Disconnected, Intermittent, and Low-Bandwidth Environments*. In Proceedings of the 2016 Symposium on Architectures for Networking and Comm. Systems, New York, USA, pp. 135–136.

[156] ter-Mors, A.; Zutt, J.; Witteveen, C. (**2007**). *Context-Aware Logistic Routing and Scheduling*. In ICAPS, pp. 328–335.

[157] Thrun, S. (**2008**). *Simultaneous localization and mapping*. Robotics and cognitive approaches to spatial mapping, pp. 13–41.

[158] Trevisani, E.; Vitaletti, A. (**2004**). *Cell-ID location technique, limits and benefits: an experimental study*. WMCSA 2004: 6th IEEE Workshop on Mobile Computing Systems and Applications, pp. 51–60.

[159] Tseng, Y.C.; Wang, Y.C.; Cheng, K.Y.; Hsieh, Y.Y. (**2007**). *iMouse: an integrated mobile surveillance and wireless sensor system*. Computer, 40(6), pp. 60.

[160] Van-Diggelen, F.; Abraham, C. (**2001**). *Indoor GPS technology*. CTIA Wireless-Agenda 2001, Dallas.

[161] Van-Setten, M.; Pokraev, S.; Koolwaaij, J. (**2004**). *Context-aware recommendations in the mobile tourist application COMPASS*. In International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Springer Berlin Heidelberg, pp. 235–244.

[162] Vera, R.; Ochoa, S.F.; Aldunate, R.G. (**2011**). *EDIPS: an Easy to Deploy Indoor Positioning System to support loosely coupled mobile work*. Personal and Ubiquitous Computing, 15(4), pp. 365–376.

[163] Vossiek, M.; Wiebking, L.; Gulden, P.; Wieghardt, J.; Hoffmann, C.; Heide, P. (**2003**). *Wireless local positioning*. IEEE Microwave Magazine, 4(4), pp. 77-86.

[164] Wang, J.; Prasad, R.V.; An, X.; Niemegeers, I.G. (**2012**). *A study on wireless sensor network based indoor positioning systems for context-aware applications*. Wireless Communications and Mobile Computing, 12(1), pp. 53–70.

[165] Wang, W.; Guo, F. (**2016**). *RoadLab: Revamping Road Condition and Road Safety Monitoring by Crowdsourcing with Smartphone App*. In Transportation Research Board 95th Annual Meeting, No. 16–2116.

[166] Wassi, G.I.; Despins, C.; Grenier, D.; Nerguizian, C. (**2005**). *Indoor location using received signal strength of IEEE 802.11 b access point*. IEEE Canadian Conference on Electrical and Computer Engineering, pp. 1367–1370.

[167] West Australia Department of Transport. (**2012**). *Planning and designing for pedestrians: guidelines*, uncontrolled copy.

[168] Xu, Y.; Yuan, H.; Wei, D.; Lai, Q.; Zhang, X.; Hao, W. (**2015**). *Research on Multi-Source Fusion Based Seamless Indoor/Outdoor Positioning Technology*. In Proc. of the China Satellite Navigation Conference (CSNC'15), Springer Berlin, 3, pp. 819–838.

[169] Yiu, T.; Hu, W.C. (**2006**). *U.S. Patent Application No. 11/385,868*.

[170] Yoon, S.; Oh, H.; Lee, D.; Oh, S. (**2011**). *Virtual lock: a smartphone application for personal surveillance using camera sensor networks*. In IEEE 17th Int. Conference on Embedded and Real-Time Computing Systems and Applications, 2, pp. 77–82.

[171] Zhang, P.; Zhao, Q.; Li, Y.; Niu, X.; Zhuang, Y.; Liu, J. (**2015**). *Collaborative WiFi*

fingerprinting using sensor-based navigation on smartphones. Sensors, 15(7), 17534–17557.

[172] Zipf, A.; Malaka, R. (**2001**). *Developing location based services for tourism: the service providers' view*.