Opportunistic computing in fully decentralized and mobile networks

SAPIENZA
UNIVERSITÀ DI ROMA

Lorenzo Bergamini

Department of Computer and Systems Science

Sapienza University of Rome

A thesis submitted for the degree of

Philosophiae Doctor (PhD)

2013 March

Abstract

The interplay of social networking platforms, information retrieval techniques and dynamic wireless communications has grown exponentially over the past few years. Together, these tools are shaping new forms of fully decentralized computation and applications. These opportunities are amplified by the relative simplicity to set up networks of small or tiny devices, such as wireless sensors and active or passive RFIDs, that allow to extract information from the environment, to process and to share it, expanding our data access and interaction capabilities and blurring the border between the digital and physical world. These new technologies coexist with traditional ones and entail a model of communication and computation that is decentralized and opportunistic in nature, in which new challenges arise and many standard assumptions do not hold. At the same time, more mature technologies, such as cellular telephony, have evolved to the point where we have small, portable, but relatively powerful devices, often endowed with sensing capabilities, that can run sophisticated applications and can exchange data in an ad-hoc fashion using wireless technologies such as Bluetooth. As a result, a number of potential applications arise, in which users interact across the digital and physical world, exchanging and processing data extracted from their local environment in real time. While a number of theoretical models of distributed computation have been proposed in the past, new efforts have been done recently in order to describe the interaction and the computing power of networks of tiny devices with limited capabilities.

One first objective of this thesis was studying methods for the opportunistic estimation of statistics of interest in fully decentralized environments and their use to implement key tools in social networking applications. Since many scenarios of interest for this work involve dynamic networks of wireless devices with limited or very limited capabilities, another direction of research in my work was to investigate the limits of theoretical models proposed to describe them. Furthermore, many opportunistic scenarios of interest involve computations over evolving networks, dynamically formed by the interaction of wireless devices carried by humans, moving in a given area of interest. Intuitively, the performance of applications in such scenarios can be greatly affected by the characteristics of the underlying evolving

networks. In order to address this aspect and to gain an initial understanding of its impact in practice, I exploited recently proposed technologies, to perform real time tracking of social interactions occurring between individuals in the physical space. This activity resulted in the collection of mobility and interaction traces in a number of indoor scenarios. In doing this, I also investigated techniques and methods to efficiently represent and to analyze evolving social network and to reconstruct them from real interaction traces.

Contents

List of Figures

1

Introduction

The attention of the research community toward social networks and social interactions has grown more and more over the past few years, mainly thanks to the huge diffusion of online social networking platforms (e.g. Facebook, LinkedIn, Google+ and so on). Understanding social dynamics and being able to predict interaction patterns can be of the utmost importance in many "social" applications, such as reputation management, recommendation systems or information sharing platforms as well as in many "classical" distributed computation applications (collaborative computation, communication in delay-tolerant networks and so on). Most studies on the dynamics of social networks have focused on the analysis of online social networks (e.g., see (1, 2)), while only a few attempts to collect data from real social interactions have been made ((3, 4, 5), mainly because of logistical difficulties (as we will discuss in details in chapters 2 and 3 of this thesis). It is possible to look at a population of individuals as a dynamic social network, where connections between members evolve and change continuously over time. In this thesis, we want to focus our attention in particular on dynamic wireless social networks, i.e. on dynamic social networks where elements of the network (with the generic word "element" we include both real persons wearing wireless devices to detect interactions and devices forming wireless networks, as described in the following) interact through wireless connections. A first important point to remark is thus to clearly understand what is the formal difference between a static and a dynamic wireless network. Since the mobility of nodes creates new issues to the communication capabilities of the network, therefore, many established concepts in static networks must be revisited for dynamic networks. The two main differences between a static and a dynamic wireless network, according to (6), are:

- In dynamic networks, it is possible that two nodes may never be part of the same connected component, but they are still able to communicate with each other within a finite time;

- In dynamic networks, while any wireless link may be (or may be assumed to be) unidirectional, the path connecting any two nodes must be regarded as directional, i.e. the fact that there is a path from node v_i to node v_j within a designated time period does not necessarily mean there is a path from v_j to v_i within the same period.

Examples of dynamic wireless networks include (but are not limited to):

- *Wireless Sensor Networks (WSNs)*: WSNs are a first example of wireless multihop networks; a WSN is a wireless network consisting of hundreds of spatially distributed autonomous devices known as nodes, with optional capability of movement (e.g. sensors attached to animals or vehicles) using sensors to cooperatively monitor physical or environmental conditions; WSN environment is fully decentralized, and each node running an application must cope with partial information (e.g. usually a node doesn't know all of the nodes forming the network, but only part of them) and hardware limitations. In addition to one or more sensors, each node in a sensor network is typically equipped with a radio transceiver or other wireless communication devices, a small microcontroller, an antenna, and an energy source, usually a battery. A sensor network usually constitutes a wireless ad-hoc network, meaning that each sensor supports a multi-hop routing algorithm (several nodes may forward data packets to the base station). Mobile WSNs represent a first example of dynamic wireless networks.

- *Mobile Ad-hoc Networks (MANETs)*: MANETs, as the name suggests, are ad-hoc networks (i.e. networks in which the nodes have self-organizing capabilities) composed of moving nodes; a well known example of MANETs is represented by VANETs (Vehicular Ad-hoc Networks), networks intended to provide communications among nearby vehicles and between vehicles and nearby fixed equipment, usually roadside equipments. The main differences between MANETs and WSNs are the hardware limitations and the power constraints: WSNs are usually intended to be used unattended (i.e. without any human intervention) in hostile environment, powered using batteries which cannot be replaced; MANETs usually don't present so strict requirements, even if a low power consumption is desirable (as an example, nodes of a MANET could use a Wi-Fi communication, while this is not possible in WSNs due to the high energy consumption of Wi-Fi).

- *Opportunistic Networks*: they are usually considered as a particular case of MANETs, but with a non-neglecting distinguishing feature: in Opportunistic Networks each node must implement a mechanism to take custody of eventual messages destined to other nodes; in other words each node in the network could be acting as an intermediate relay during the network lifetime. In opportunistic networks communication links are usually directional, highly variable and highly unstable, so that it is not possible to assume a fixed routing infrastructure or to

adopt a routing protocol for communication: every time a node has a "chance" to communicate to a suitable neighbor [1] it will send the message.

We will focus our attention in particular on dynamic wireless networks of tiny devices characterized by ***Opportunistic*** as a first keyword; typically, we think of these networks as being large and fully decentralized, so that each node can have an accurate view of only its local vicinity; furthermore, as these networks change over time, as nodes join, leave, and move around, the neighborhood of each node changes continuously, so that it is not possible to assume any fixed routing infrastructure. In this sense, a key aspect to consider is the model of mobility regulating interactions of the devices forming the network currently monitored/-analyzed. Even if the focus of this thesis is not on mobility models, an overview of existing representations is required and will be given in chapter 2. The adopted mobility model has a strong impact on the overall behavior of the network so we believe that an exhaustive review of some classical mobility models commonly accepted by the research community and adopted in several simulation studies to validate new protocols/algorithms and to compare them with real traces of movement collected from real populations is needed. We are particularly interested in analyzing the social aspects related to the movement of persons. To this end, in chapter 2 we will review the main mobility models, discussing their main characteristics.
Despite of the adopted communication technology (Zigbee, Bluetooth, Wi-Fi and so on), dynamic wireless networks are usually composed of tiny devices with limited computational power and hard energy constraints (wireless devices are, in the vast majority of cases, battery-powered), and these aspects must be considered when realizing a protocol for communication or a theoretical model: said differently, ***Efficiency*** is the second keyword describing the type of dynamic wireless networks we are considering. In light of this, we want to point out in this thesis that even though the most various solutions, applications, protocols and so on can be realized to perform the most different tasks (area monitoring, information dissemination, viral advertising etc.etc), it is possible to consider a set of basic computation primitives that are of primary importance for many of the possible applications. Chapter 4 will consider some problems and applications in fully decentralized environments, under different constraints about communication and computational capabilities and resources of the nodes (e.g. storage, energy and so on). We will consider a very basic theoretical model for distributed computation (*Population Protocols*), and we will analyze and show how theoretical models could work fine even in a realistic context like the ones we are considering. By means of simulations and real testbeds, we evaluated the behavior of these models, and we paved the way for possible future developments of concrete applications

[1] Here intended as a neighbor that is estimated to be closer than the current node to the final destination of the packet according to some protocol-specific metrics

exploiting them. The structure of this work is thus the following. In chapter 2 we review and discuss the main existing theoretical models for generating and evaluating dynamic graphs. In the same chapter, we review some of the existing mobility models for dynamic wireless networks and we discuss the differences between using synthetic traces and real traces. In chapter 3 we focus our attention on technologies used to record mobility traces: we present some of the currently adopted technologies to collect different kinds of real traces, focusing on their advantages and limits. We conclude the chapter presenting the infrastructure we used for collecting the traces we used for our experiments and the different digital formats we adopted for representing such traces, explaining why we used different formats. In chapter 4 we evaluate the performance of algorithms to perform basic primitives in the Population Protocol model of distributed computation, but using real traces to generate interactions among nodes instead of theoretical mobility models. We will also describe in details how we implemented the population protocols model on real physical devices. We will also show techniques to perform some complex monitoring (e.g. distinct counting) and mining tasks (e.g. link prediction) in a fully distributed way. In chapter 5 we conclude the thesis, and we present some open problems and future development of our ideas.

2

Dynamic Social Networks and Mobility

We introduced the concept of mobile dynamic social networks. We define this kind of networks as a special case of dynamic networks, where the communication is wireless and opportunistic, in the sense that it happens on the basis of the social interactions between members of the population. Dynamic networks and systems have always been present in nature, thus, an appropriate modeling and analysis of these real models is a perfectly natural process. An interesting though non-exaustive set of examples of real dynamic networks is presented in (7):

- Citation Networks: this is the network of scientific papers' citations. The graph representing this continuously evolving network is extremely simple: a new vertex is added whenever a new paper is published; new edges are added between the new vertex and the papers it cited.
- Collaboration Networks: real-existing and well studied examples of this type of networks are IMBD (Internet Movie Database (8)) or DBLP (Computer Science Bibliography (9)); in both cases, a new edge is added when a new author/actor participates to a new paper/movie, and edges are added to existing authors/actors with which the new vertex collaborated. If all actors/authors of a new work are already present in the graph, a new edge between them is added, or the weight of an already existing link is increased.
- Communication Networks: an example of a communication network is for example the internet. A vertex represents a computer connected to the internet (a personal computer, a web server...); an edge is added every time a connection is established between one computer and another (e.g. user A visits website B, then an edge A-B is added to the dynamic graph representing the network).

These examples represent networks characterized by a slow variation in time. Our interest is mainly on dynamic networks where the variability is very high, as we will show in the following. The most adopted way to represent (static) networks is in general through the use of graphs $G(V, E)$, where V represents the set of vertices (i.e. the elements belonging to the network) and E represents the set of edges (i.e. the connections between vertices). In dynamic networks both the set of vertices V and the set of edges E can change over time.

Definition 1 *A Graph is defined as $G = (V, E)$, where V is the finite set of vertices (i.e. elements of the graph), while E is the finite set of edges. A particular vertex i is indicated as v_i, while a particular edge connecting 2 vertices i and j is indicated as $e_{i,j}$.*

A graph may be *bidirectional*, when there is no difference between an edge $e_{i,j}$ and an edge $e_{j,i}$, or *directional* where an edge $e_{i,j}$ is different from an edge $e_{j,i}$ A *Vertex Weighted Graph* is a graph where a function f is defined on the vertices as $f : V \to N$; f represents the *Weight* of a vertex. Similarly, an *Edge Weighted Graph* is a graph where a function g is defined on the edges as $g : V \to N$; g represents the *Weight* of an edge.

Properties of Graphs Static graphs are usually evaluated on the basis of some properties, as we will explain below, reporting the list of properties described in (7).
Degree. The degree k of a particular vertex v is defined as the total number of its edges; if considering a directional graph, then the degree k is defined as $k = k_{in} + k_{out}$, where k_{in} is called the incoming degree and is defined as the total number of incoming edges, while k_{out} is called the outcoming degree and is defined as the total number of outcoming edges. In the analysis of a graph, the degree distribution is generally considered.
Clustering Coefficient. It is a measure of the degree to which nodes tend to cluster together. A very good definition was given by Watts and Strogatz in (10): "The clustering coefficient C is defined as follows. Suppose that a vertex v has k_v neighbours: then at most $k_v(k_v - 1)/2$ edges can exist between them (this occurs when every neighbour of v is connected to every other neighbour of v). Let C_v denote the fraction of these allowable edges that actually exist. Define C as the average of C_v over all v. For friendship networks, this statistics has an intuitive meaning: C_v reflects the extent to which friends of v are also friends of each other, and thus C measures the cliquishness of a typical friendship circle."
Shortest Path. In non-weighted graphs, the shortest path is defined as the minimum number of edges to cross for reaching a vertex v_j starting from a vertex v_i. The average shortest path is defined as the average of all the shortest paths

between every possible couple of vertices in the set V.
So far we focused our attention on static graphs, i.e. on graphs that do not change over time. Another important class of graphs is that of *Dynamic Graphs*. Several different theoretical models have been proposed in literature to generate dynamic graphs, each of them with its own peculiar characteristics. A possible classification of *Dynamic Graphs* is based on the change of one or more of the following properties (11):

- Node dynamic, where the vertex set V changes over time;
- Edge dynamic, where the edges set E changes over time;
- Node weight dynamic, where the function f varies over time;
- Edge weight dynamic, where the function g varies over time.

As for the dynamic graphs we consider in our work, the 4 properties may change all together in different time intervals. The variation of the 4 properties above raises a class of graphs usually indicated as *Evolving Graphs*. Throughout this thesis we will adopt the following definition, taken from (12) (and similar to the one given in (13)):

Definition 2 *Let $\mathcal{G} = G_1, G_2, ...$ be an infinite sequence of graphs on the same vertex set V. We call this sequence an Evolving graph. We say that $\mathcal{G}$ has the graph property X if every graph G_i in the sequence has the property X.*

It is possible to evaluate evolving graphs using the same properties we introduced above for static graphs, but it is important to underline that, given the definition 2, the property must be valid for all the graphs forming the sequence. An example illustration of a generic evolving graph is depicted in figure 2.1, while in figure 2.2 we reported an alternative representation, where the number on each edge indicates the time interval presence of the given edge.

In the opening of this chapter, we reviewed a few examples of dynamic networks taken from real life. The important thing to underline is that in the examples we presented the variability in the networks is quite slow, and they do not take into account the impact of mobility on the resulting dynamic graph, since they do not represent populations of moving individuals. Mobility is another ingredient we are interested in considering. Recall that our attention is focused on mobile dynamic social networks, i.e. on dynamic networks where interactions (and thus the corresponding edges in the dynamic graph) happen depending on the movement of the elements of the network. It is evident how 2 different mobility models applied to the same population, can originate completely different evolving graphs, both in terms of graph sequence and structural properties, thus it is extremely important to know in details the characteristics of the adopted model. Following

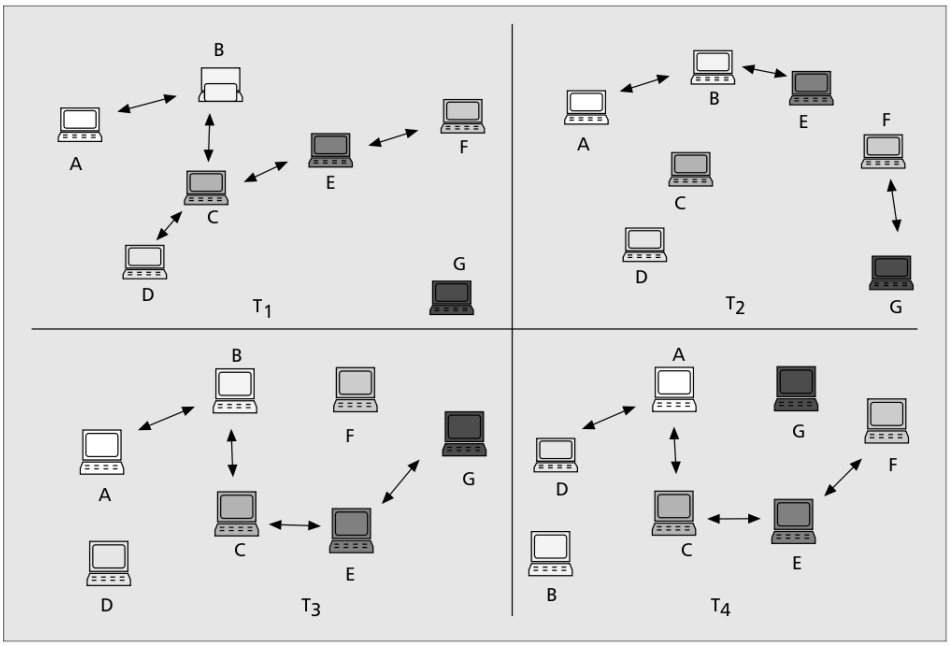


Figure 2.1: Example of an evolving graph. Indices corresponds to successive snapshots

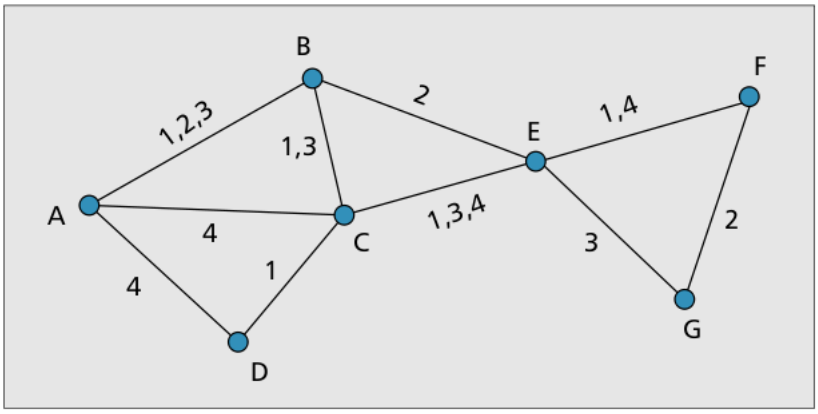


Figure 2.2: Alternative graphical representation of the graph in figure 2.1. Edges are labeled with corresponding presence time interval

this idea, we will discuss in the following some mobility models proposed in literature, focusing on the reasons why they have been adopted (or not) in previous works, and spending a few words on the reasons why they fit our needs (or not). In chapter 4 we will present our experiments and we will motivate the use of a particular mobility model.

2.1 Graph Generative Models

This section will be devoted to a description of several theoretical models for generating random graphs. Recall from the introduction that the main goal of the thesis is to analyze the possibility of computations in real dynamic social networks, and to show that implementing in practice some theoretical models or problem is possible. Here we are interested in reviewing some of the classical models, commenting if they are appropriate or not for our mobile dynamic social networks.

2.1.1 Erdös Renyí random graph model

This is probably the grounding model for the analysis of dynamic networks. It was introduced in 1960 in the seminal paper (14) and it is currently used as a possible representation of random graphs. According to the Erdös Renyí (ER) model, a random graph is built by picking N nodes, and for each couple of nodes, an edge E is added with a given probability p. Thus, an ER random graph is defined as $G_{(N,p)}$, where N is the number of vertices and p the probability of a given edge to exist. An example of ER graphs with $N = 20$ and 3 different values of probability p is depicted in figure 2.3

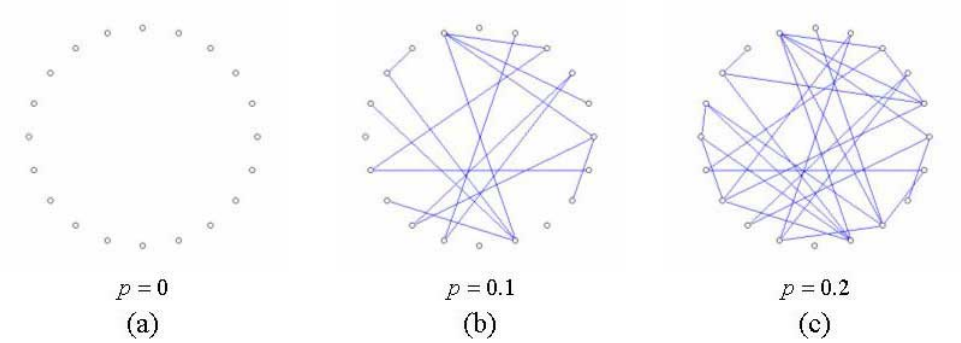


Figure 2.3: Example of ER graphs when varying p

It was shown that the degree distribution of ER random graphs follow a Poisson distribution as the number of nodes in the network goes to infinity. This classical

model, although simple in its basic assumptions, reflects in a certain way the structure of the dynamic social networks we observed during the collection of real data (see chapters 3 and 4 for additional details). It is important to underline, in any case, that the ER model is not a pure dynamic model: it is a probabilistic model than *can* be used to generate dynamic graphs in the sense of definition 2, when applied repeatedly. In this case, even if in the real dynamic social network interactions (which can be seen as edges) between persons (seen as nodes with no loss of generality) are based on their movement, in a given way it is possible to review such interactions as ruled by a probability. So, the classical ER model well reflects, even if with some limitations, the behavior of a real dynamic social network.

2.1.2 Edge-Markovian Evolving Graphs

Similar to the original Erdös Renyí model, but different in the fact that it actually generate dynamic graphs, this model was introduced in (15) . The authors introduce a stochastic time-dependency in the ER model. Starting from an arbitrary initial edge probability distribution, they assume that at every time step every edge exists or not following a two-state Markovian process, with probabilities p (called *edge birth-rate*) and q (*edge death-rate*). In other words, if an edge exists at time t then, at time $t+1$, it dies with probability q. If instead the edge does not exist at time t, then it will come into existence at time $t+1$ with probability p. This model was introduced by the authors to investigate the speed of information dissemination in this kind of dynamic graphs. As for the standard ER model, the Edge-Markovian model is suitable to represent a dynamic social network. It is even better than the standard ER model, because it is able to catch the strong dynamicity of a real population of persons. In a dynamic social network of real individuals, where the aim of an experiment is for example to collect data on conversations between persons, the use of 2 probalities is perfectly suitable to model, for example, the beginning of a new conversation (edge birth-rate) or the end of a conversation existing on a previous time interval (edge death-rate).

2.1.3 T-interval connectivity model

This model was first presented in (16); the authors consider a dynamic graph composed of devices with an identifier and were the *T-Interval Connectivity* property holds. This property is defined as follows

Definition 3 *We say a dynamic graph $G = (V, E)$ is T-Interval Connected for $T \geq 1$ if for all $r \in \aleph$, the static graph $G_{r,T} = (V, \bigcap_{i=r}^{r+T-1} E(r))$ is connected.*

The graph is said to be ∞-interval connected if there is a connected static graph $G' = (V, E')$ such that for all $r \in \aleph$, $E' \subseteq E(r)$

In other words, the authors assume that for every T consecutive rounds there exists a stable connected spanning subgraph in the considered dynamic graph. This model is used by the authors to present and analyze some classical problems of distributed computing (details will be given in chapter 4). The T-interval connectivity model is not completely suitable for our mobile dynamic social networks, since the mobility of users make it unrealistic to set a connectivity interval. Nonetheless, in order to accurately detect a contact, it could be useful to use this property during the post-processing of traces to filter out false contacts or contacts happening by chance. One could assume, for example, that a contact is valid if and only if for at least t successive timestamps, a neighbor node is in the list of contacts of the other node.

2.1.4 Watts-Strogatz Graph Model

In (10) Watts and Strogatz introduced a new model for generating random graphs, mainly to overwhelm 2 limitations of the classical ER-model:

- ER graphs have a low clustering coefficient, since the probability of two nodes being connected is constant, random and independent.
- ER graphs are not able to catch some real world phenomena, like for example the formation of hubs. The degree distribution of ER graphs, in fact, converges to a Poisson distribution, while many real-world networks have been observed to convervge to a power law distribution.

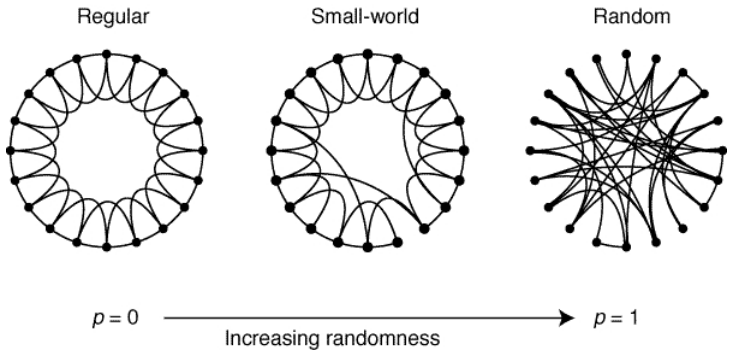


Figure 2.4: Example of WS graphs

The Watts-Strogatz (WS) model uses a procedure to generate the random graph based on a rewiring procedure presented in (10) that aims to analyze the behavior

of networks when the probability of having an edge between a couple of nodes varies between 0 and 1. Watts and Strogatz consider a ring lattice with n vertices and k edges per vertex as a starting point, and they assume each edge to be rewired at random with probability p (see figure 2.4, taken from the original paper). Assuming that the following condition holds: $N >> K >> ln(N) >> 1$, then for every node $n_i = n_0, ..., n_{N-1}$, take every edge (n_i, n_j) with $i < j$ and rewire it with probability p. The rewiring procedure is done by replacing the edge (n_i, n_j) with (n_i, n_k), where k is chosen with uniform probability from all the possible vertices that avoid self-loops and link duplication.

2.1.5 Barabasi-Albert Graph Model (Preferential Attachment)

This new model was presented for the first time in (17) and is usually referred to as Barabasi-Albert (BA) model, from the names of the 2 authors. By analyzing several large databases containing information on citations patterns or properties of the WWW (in other words: information of different types of dynamic networks), they found out that independently of the system, the probability p of a vertex interacting with other k vertices follows a power law distribution, namely $p \approx k^{-\gamma}$. Their novel conclusion was then that large networks tend to self-organize in a scale-free state, and they introduced the concept of "Preferential Attachment" and of "growth" to create a new random graph model that better reproduces the behavior of real dynamic networks. The main assumptions of the previous models that Barabasi and Albert discussed are the fact that in both ER and WS models, the graph starts with a fixed number N of vertices that are randomly connected (ER) or rewired(WS). They observed that in real networks, we experience the continuous addition of new vertices to the network itself, that are usually connected to existing vertices. The second assumption "criticized" by Barabasi and Albert is that the probability of connecting 2 vertices is random and uniform, while they observed the existence of a preferential connectivity in real networks. In other words, the probability with which a new vertex connects to existing vertices is not uniform, but it depends on the number of connections existing vertex already has (the more connections, the greater the probability to receive a new connection). The new model, then, starts with a small number of vertices (m_0) and at every time step, a new vertex with $m(\geq m_0)$ edges linking it to m different vertices is added to the graph. To take into account preferential attachment, the probability P that a new vertex is connected to existing vertex i depends on the connectivity k_i of that vertex, so that $P(k_i) = \frac{k_i}{\sum_j k_j}$. After t time steps, the model leads to a random network with $t + m_0$ vertices and mt edges. An example of a graph growing according to the model by Barabasi and Albert is depicted in figure 2.5.

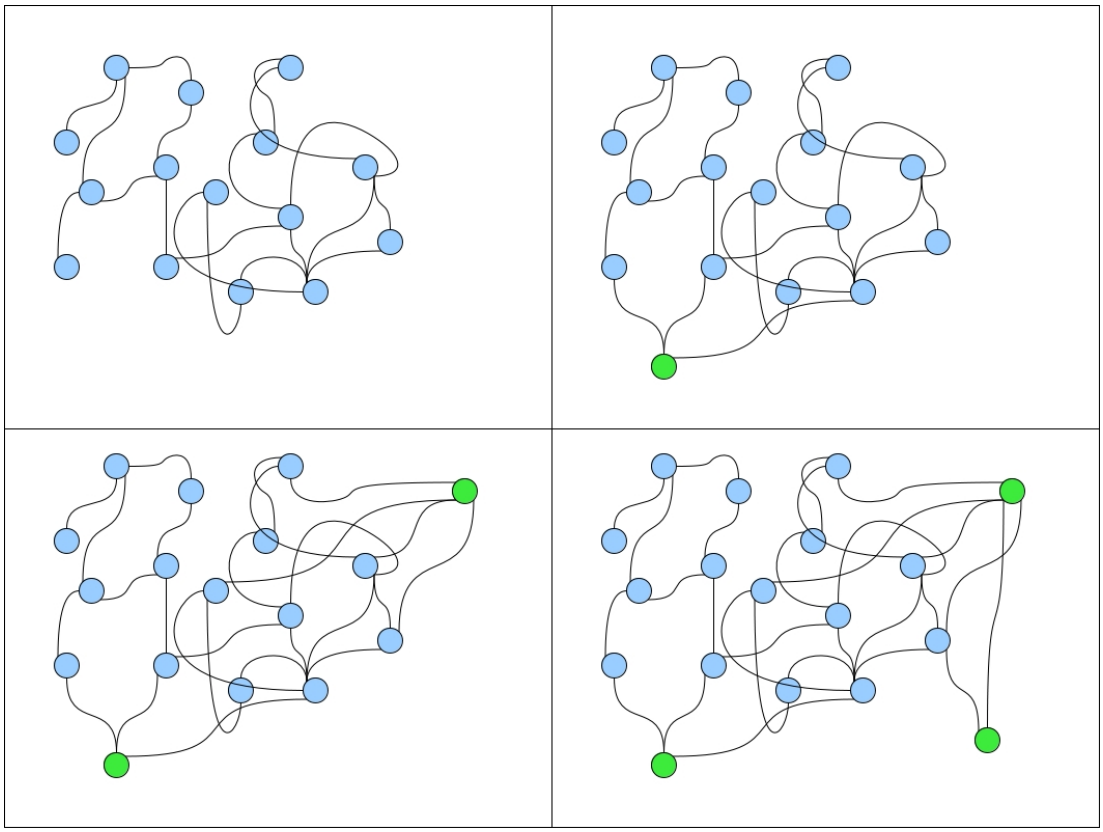


Figure 2.5: Example of a BA graph

It is important to remark, anyways, that the BA model is not suitable for modeling our dynamic social network. In the context we are considering, the number of nodes is set at the beginning of the experiment and it is not expected for new nodes to join the network during the experiment. It would be extremely interesting to study the behavior of persons when a clear difference is established at the beginning of the experiment (for example, considering the experiment at the museum we will describe in chapter 3, it would be nice to analyze if the population shows a preferential attachment behavior when sensors are given to museum guides too, and not to "normal" visitors only). After a "training" period, a new group of persons could be invited to join the experiment, and we could observe if they tend to preferentially join museum guides or not. But for now, this experiment is left as a future work. We will simply state that for the experiment we conducted so far, the preferential attachment model does not fit properly our envisioned dynamic social network.

2.1.6 Kronecker Graphs

This alternative technique for graphs generation was introduced in (18) and (19) to capture the shrinking diameter property and the densification power law in a way simple to be analyzed. The author, in fact, observed that networks tend to become denser over time, according to $E(t) \propto N(t)^a$, where a is the densification exponent (with $1 \leq a \leq 2$), E is the number of edges and N is the number of nodes. They also observed that as the network grows, the distances between nodes tend to slowly decrease. The novel idea was to generate graphs recursively. To realize their intuition, the authors started from the classical Kronecker product of matrices, defined as follows:

$$C = A \otimes B = \begin{pmatrix} a_{1,1} & a_{1,2}B & ... & a_{1,m}B \\ a_{2,1} & a_{2,2}B & ... & a_{2,m}B \\ ... & ...B & ... & ... \\ a_{n,1} & a_{n,2}B & ... & a_{n,m}B \end{pmatrix}$$

and introduced the Kronecker product of two graphs (18) defined as the Kronecker product of their adjacency matrices. At this point, iterating the Kronecker product as follows $G_k = G_1 \otimes G_1 \otimes .... \otimes G_1$, we obtain a sequence of graphs, where at each Kronecker multiplication the size of the graph increases exponentially. At this point, in fact, G_k has N_1^k nodes and E_1^k edges, recreating the *densification* property. The author proved in (18) that Kronecker graphs have the following structural properties: for the static network, they follow a power law degree distribution, a power law eigenvalue and eigenvector distribution and the small diameter; for the dynamic network they capture the densification power law and the shrinking/stabilizing diameter properties. For the detailed mathematical demonstrations please refer to the original paper we cited.

2.2 Mobility and Dynamic Networks

We described so far some theoretical models currently used to generate dynamic networks. A dynamic mobile social networks, however, is composed of a population of individuals where interactions between one user and another is regulated by their movement inside the area of interest. We can say that two users i and j are interacting at time t (i.e. an edge (i, j) appears in the dynamic graph representing the dynamic network under investigation at time t) if, during their movement, they come sufficiently close to communicate. Details on the granularity needed to infer an ongoing social relationship will be given in chapter 3; for now we want to introduce the topic of mobility models, whose adoption strongly influences the interactions happening in the mobile dynamic social network under investigation. The choice of a particular mobility model to simulate and evaluate the performance of a communication protocol on dynamic wireless networks has a strong impact on the overall results, since interactions may vary significantly from one model to the other. Several attempts have been made to implement different models able to capture the real mobility patterns of a given population (being it humans, vehicles, animals or whatever else), following essentially 2 approaches: using synthetic models or real traces. Synthetic models are purely mathematical models which try to reproduce the movement according to specific rules (e.g. equations) which vary from model to model, and a non-exhaustive survey of the most known and adopted mobility models will be presented in this section. Real traces are real mobility traces collected by a population of physical individuals (or vehicles, or animals) by means of data logs (GPS traces, bluetooth logs, WiFi traces and so on), and then used to feed a simulator. While real-world traces give a realistic representation of some real systems, synthetic models only approximate this behaviour. The main advantage of using synthetic models is in the fast and easy generation; furthermore it is usually possible to fastly modify some parameters to evaluate the behavior of a given algorithm in several different scenarios, thus obtaining indications on its strengths and weaknessess in different conditions (e.g. the tested algorithm performs very well in dense networks, and performs poorly in sparse networks). The main disadvantage of using synthetic models, however, is that in most cases they do not reproduce the realistic conditions under which the algorithm is intended to work, as the resulting movement tend to be quite far from reality, as confirmed by an analysis of some real traces (20). Nonetheless, simulations using synthetic models are useful to obtain important indications, at least in a preliminar phase of testing. Collecting real traces from the real world, on the other hand, is a non-trivial and time-consuming task, as a suitable number of persons must be involved in the experiment, and an appropriate infrastructure for data collection must be installed. Post-processing collected traces and representing them in a digital form that a simulator can easily understand is another time-consuming task. The main advantage is that real traces

allow an accurate analysis and simulation of what the behavior of the algorithm should have been when considering real interactions between real elements (we are still using the generic word *element* to remark the fact that the distinction between synthetic and real traces can be applied to every simulation context, and it is not limited to our experiments focused on social interactions). One can thus expect a greater adherence to reality when using real traces to evaluate an algorithm. In the last few years, several different technologies were adopted to collect data from real groups of persons, and chapter 3 will be completely focused on the description of the most adopted technologies and their limits, and on the issues of different digital formats for representing the collected traces. A third approach to the realization of mobility models is an hybrid one: designing synthetic mobility models using real traces as a starting point to extract useful mathematical indications on the movement of the population. To classify the models in the following description, we used the same taxonomy adopted in (21): we distinguish between purely synthetic mobility models and trace-based mobility models. An additional division will be made between models applied to each single element of the network independently from the others, and models where the movement of the elements depend on the movement of other elements (so-called "Group Mobility Models").

2.2.1 Purely Synthetic Single Node Mobility Models

In this section we present some of the most diffused synthetic mobility models. We already remarked why using a synthetic model can be useful for a preliminary evaluation of a given protocol under several different conditions, but we also pointed out the limits of synthetic models, mainly in terms of unrealistic movement. It is usually possible to tune several parameters of a synthetic model, ranging from the velocity of a node, to the duration of a given movement direction, to the dependency on previous decisions and many others. Obviously, having too many parameters to tune affects the usability of the model itself, thus a good synthetic model must be as simple as possible, but at the same time it must provide users the possibility to test their solutions in several different conditions. The main strength of synthetic models is in the possibility of generating several different scenarios, while the use of real traces binds the user to a single scenario (at least for what concerns movement and interactions. Many other simulation parameters could be varied, as we will discuss in chapter 3).

2.2.1.1 Random Waypoint Model

The Random Waypoint model was proposed for the first time by Johnson and Maltz (22). Depending on the specific implementation of the adopted simulator,

some details of the model may be slightly different from one solution to the other; we will refer to the implementation used in NS-2 simulator, probably the most diffused (23). The behavior of this model is extremely simple: when starting the simulation, every mobile node picks a random point D of the simulation field as its destination, and starts moving toward D with a constant speed V uniformly and randomly selected in the interval $[0, V\_max]$, where V_max is a parameter specified by the user and indicating the maximum speed allowed for each node in the current simulation. When the node reaches its destination D, it stops for a given time interval, after that it selects a new random destination D' and starts moving toward it with a new uniform velocity V'. The whole process is repeated until the simulation ends. We remark the lack of any correlation between the various destinations and between the various velocities; this takes to unrealistic movements like sharp turns or sudden accelerations/decelerations. Despite of its unrealistic behaviour, random way point model is still the most adopted synthetic model (23) for simulations. This is due to its simplicity and to its few parameters to set. Nonetheless, an algorithm performing well with this model cannot be assumed to work fine when applied to a more realistic model of movement, since as we already remarked, random way point is quite far from reality. Nonetheless, since random way point model is commonly accepted by the scientific community, we decided to use it in our experiments (see chapter 4 for details).

Random Walk Model A slight extension of the Random Waypoint model is the Random Walk Model. The difference between the two is in the fact that the stop time of a node when reaching its destination is, in this second case, 0 and in the fact that in the Random Walk each node changes its parameters (final destination, i.e. direction of movement, and uniform velocity) at every time interval t, where t is a parameter specified by the user, while in the previous model parameters vary only when a node actually reaches its final destination. As in the previous case, there is no correlation between the parameters chosen by a mobile node for a given time interval and those adopted by the same node during the previous intervals.

Random Direction Model An interesting problem with both Random Waypoint and Random Walk models was observed by Bettstetter (24) and by Blough et al.(25) and well described in (23): "The spatial node distribution of Random Waypoint model is transformed from uniform distribution to non-uniform distribution after the simulation starts. As the simulation time elapses, the unbalanced spatial node distribution becomes even worse. Finally it reaches a steady state. In this state the node density is maximum at the center region, whereas it is almost zero around the boundary of the simulation area. [...] This phenomenon results from the certain mobility behavior of Random Waypoint model: since the

nodes are likely to either move towards the center of simulation field or choose a destination that requires movement through the middle, the nodes tend to cluster near the center region of the simulation field and move away from the boundaries". The Random Direction Model was proposed by Royer et al. (26) to overwhelm this limitation. Instead of selecting a random destination within the simulation field, in this model the node randomly and uniformly chooses a direction by which to move along until it reaches the boundary; when reaching the boundary the node stops with a pause time and then chooses another direction to travel. This way, nodes remain uniformly distributed within the simulation field. The drawback is that it is not realistic at all to assume a single continuous direction of movement until reaching the simulation boundaries.

Smooth Random Mobility Model To avoid the sharp turns and sudden accelerations typical of Random Waypoint model, Bettstetter in (27) proposes to change the speed and direction of node movement incrementally and smoothly. It is observed that mobile nodes in real life tend to move at certain preferred speeds $\{V^1_{pref}, V^2_{pref}, V^3_{pref}, ..., V^n_{pref}\}$ rather than at speeds purely uniformly distributed in the range $[0, V_{max}]$. Therefore, in Smooth Random Mobility models, the probability distribution of node velocity is as follows: the speed within the set of preferred speed values has a high probability, while a uniform distribution is assumed on the remaining part of the entire interval $[0, V_{max}]$. The frequency of speed change is assumed to be a Poisson process; upon an event of speed change, a new target speed $v(t)$ is chosen according to the given probability distribution function of speed and the speed of the mobile node is changed incrementally from the current speed $v(t')$ to the targeted new speed $v(t)$ by acceleration or deceleration speed $a(t)$. The probability distribution function of acceleration or deceleration is uniformly distributed among $[0, a_{max}]$ and $[a_{min}, 0]$ respectively. Thus, for each time slot t, the new speed is calculated as $v(t) = v(t - \Delta t) + a(t)\Delta t$. Unlike speed, the movement direction is assumed to be purely uniformly distributed in the interval $[0, 2\pi]$. Once a movement direction is chosen, the node moves in a straight line until the direction changes. The frequency of direction change is assumed to have an exponential distribution in (27).

2.2.1.2 Gauss-Markov Mobility Model

The Gauss-Markov model was originally proposed by Liang and Haas in (28) and it was designed to adapt to different levels of randomness via one tuning parameter. In this model the velocity of the mobile node is assumed to be correlated over time and modeled as a Gauss-Markov stochastic process. At the beginning of the simulation, a speed and a direction are assigned to each node, and at fixed time intervals these values are updated on the basis of the previous instant according

to the following equations (29):

$$s_n = \alpha s_{n-1} + (1-\alpha)\overline{s} + \sqrt{(1-\alpha^2)}s_{x_{n-1}}$$
$$d_n = \alpha d_{n-1} + (1-\alpha)\overline{d} + \sqrt{(1-\alpha^2)}d_{x_{n-1}}$$

where s_n and d_n are the new speed and direction of the mobile node at time interval n and α is the tuning parameter used to vary the randomness; $\overline{d}$ and $\overline{s}$ are constants representing the mean value of speed and direction and $s_{x_{n-1}}$ and $d_{x_{n-1}}$ are random variables from a Gaussian distribution. Totally random values are obtained by setting $\alpha = 0$ and linear motion is obtained by setting $\alpha = 1$. Intermediate values are obtained by setting $0 < \alpha < 1$, with greater α values implying a major influence of past velocities on the current value. An example path followed by a node moving according to this model is reported in figure 2.6

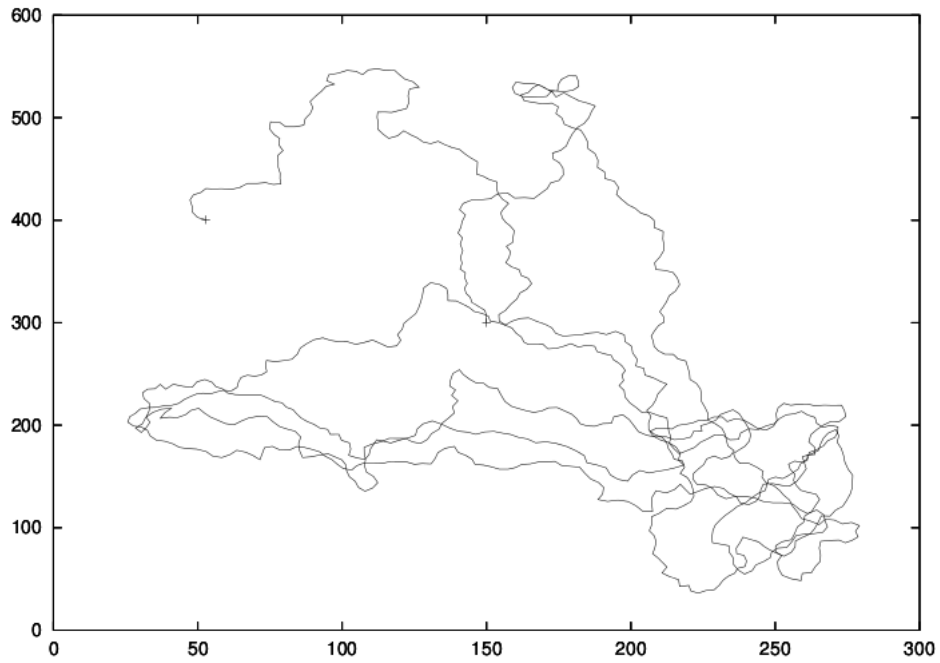


Figure 2.6: Example of the pattern followed by a node moving according to Gauss-Markov model

2.2.1.3 Geographic Mobility Models

The idea behind geographic mobility models is to imitate the real conditions in which a population is expected to move, by recreating physical obstacles like

buildings or walls. This class of mobility models is quite interesting, even if the growing complexity of a detailed modeling like this does not correpsond to a growing accuracy of the simulated results. Furthermore, obstacle mobility model takes into account also effects of obstacles on the propagation of the radio signal and this element adds complexity to the realization of the model. We believe that a mobility model should only model the real movement of objects, while other elements like radio propagation or communications should be taken into account by the adopted simulator (we will go back to this point in chapter 3, while presenting our analysis of network simulators). Nonetheless, it is important for a detailed survey, to present these solutions, even if we didn't considered them in our experiments.

Pathway Mobility Model The basic idea of this model, introduced in (30), is to model the simulation field (the authors focus on the map of a city) like a graph, either randomly generated, or carefully modeled on the basis of the map of a real city. The idea is that vertices in the graph represent buildings, while edges represent streets and connections between buildings. At the beginning of the simulation, each node is placed on a random edge and it chooses a random destination where it starts moving through the shortest path following the edges. When the destination is reached, similarly to the Random Waypoint model, the node waits for a short time, then it picks another random direction and starts moving again. The process is repeated until the simulation ends.

Obstacle Mobility Model These models were first proposed in (31). Models belonging to this category starts from the observation that when deploying mobile networks in real scenarios, many concrete obstacles both in indoor (walls, furniture, persons) and outdoor (buildings, vehicles and so on) settings influence both the movement of the user and the propagation of the radio signal. In their work the authors modeled 3 different scenarios; in all of them some obstacles are randomly placed in the simulation field and each node must calculate its trajectory in order to avoid the obstacles. If an obstacle is placed between 2 nodes, they cannot communicate until one exits from the obstructed area. A similar investigation, with restricted radio propagation and movement in presence of obstacles was made in (32).

2.2.1.4 SWIM - Small World In Motion

This mobility model was firts presented in (33). It is intended to model the movement of real persons, and it arises from the following observations:

– a location is usually chosen using a trade-off: popularity vs distance from home;

- there are a few places where persons spend a long time (office, home)
- the speed at which we move toward a destination is proportional to its distance (the farther the destination, the higher the speed; a very far destination is usually reached using a vehicle)

On the basis of these intuitions, the authors created a very simple mobility model, where all the above mentioned observations are taken into account. Essentially, each node at the beginning is assigned a location named *home*, i.e. a randomly and uniformly chosen point over the simulation area. At each step, the node assigns a *weight* to every possible destination; this value increases with the popularity of the place and decreases with its distance from the node's home. When the node needs to move, it chooses its next destination randomly and proportionally with its weight. For additional details, please refer to the original work in (33). For the purpose of this thesis it is sufficient to remark that the experiments made by the authors using real traces available online confirmed that the SWIM model respects the power-law distribution observed in real human groups.

2.2.1.5 Mobility Models based on Social Network Theory

The models we presented so far, in some cases miss the social aspect of movement. Mobility models are usually adopted to simulate the behavior of mobile ad-hoc networks, and a lot of times ad-hoc networks are composed of humans carrying hand-held devices, like smartphones or tiny sensors (the term Pocket-Switched Networks was introduced by Hui et al. in (5) to refer to this type of networks) and the movement of humans is regulated by their social relationships. Thus, to better model the real movement, it is fundamental to take into account human behavior. For this reasons, a couple of models based on this observation have been proposed; a detailed description of the most important (to the best of our knowledge) is given in the following.

Community Based Mobility Model This model was presented in (34), but it can be considered as an evolution of the basic model first presented in (35). The authors propose a model in which some characteristics of social network theory are considered. Essentially, the procedure they followed to develop their model can be summarized as follows:

- Modeling a social network, and using the social network structure as input of the mobility model
- Establishing the mobility model
- Defining the dynamics of the nodes

The first step is realized by means of a so-called *Interaction Matrix*. Considering the social network in figure 2.7 [1]

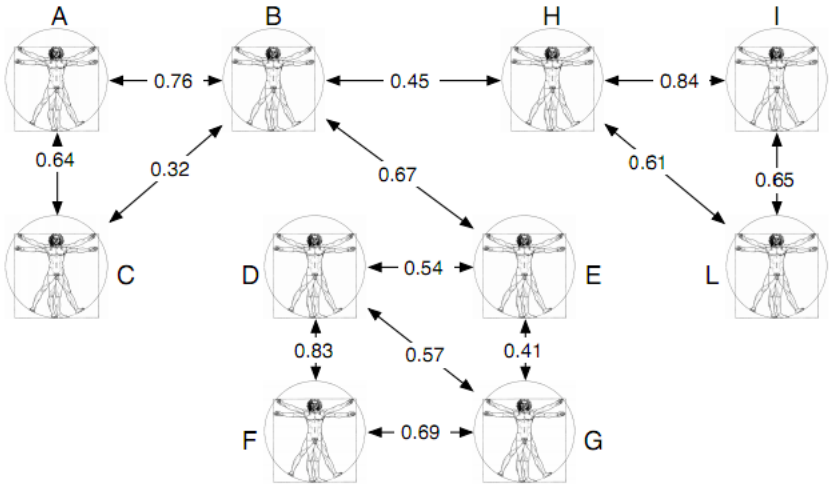


Figure 2.7: An example social network

its corresponding interaction matrix is this one

$$\mathbf{M} = \begin{bmatrix} 1 & 0.76 & 0.64 & 0.11 & 0.05 & 0 & 0 & 0.12 & 0.15 & 0 \\ 0.76 & 1 & 0.32 & 0 & 0.67 & 0.13 & 0.23 & 0.45 & 0 & 0.05 \\ 0.64 & 0.32 & 1 & 0.13 & 0.24 & 0 & 0 & 0.15 & 0 & 0 \\ 0.11 & 0 & 0.13 & 1 & 0.54 & 0.83 & 0.57 & 0 & 0 & 0 \\ 0.05 & 0.67 & 0.24 & 0.54 & 1 & 0.2 & 0.41 & 0.2 & 0.23 & 0 \\ 0 & 0.13 & 0 & 0.83 & 0.2 & 1 & 0.69 & 0.15 & 0 & 0 \\ 0 & 0.23 & 0 & 0.57 & 0.41 & 0.69 & 1 & 0.18 & 0 & 0.12 \\ 0.12 & 0.45 & 0.15 & 0 & 0.2 & 0.15 & 0.18 & 1 & 0.84 & 0.61 \\ 0.15 & 0 & 0 & 0 & 0.23 & 0 & 0 & 0.84 & 1 & 0.65 \\ 0 & 0.05 & 0 & 0 & 0 & 0 & 0.12 & 0.61 & 0.65 & 1 \end{bmatrix}$$

Figure 2.8: The interaction matrix for the above social network

Each element $m_{i,j}$ of the matrix defines the strength of the interaction between the two individuals i and j are named *interaction indicators*. From this matrix, the authors derive the so-called *Connectivity Matrix*, which is a binary matrix where a 1 is placed as an entry $c_{i,j}$ if and only if $m_{i,j} > t$ where t is a threshold parameter arbitrary chosen. The Interaction Matrix (and consequently the Connectivity Matrix) can be derived from a mathematical model, from a sociological investigation, or even from real interaction traces. Once the two matrices have

[1] The figure is taken from the original work in (34)

been defined, the authors used an algorithm to detect the presence of communities is social networks (for details on this algorithm, please refer to (36)); we omit a detailed description of the algorithmic procedure, it is here sufficient to say that applying the algorithm to our social network of example, 3 communities are detected: $C_1 = A, B, C, C_2 = D, F, G, g, C_3 = H, I, L$. After the communities are identified, they are randomly associated to a location on the simulation area (the authors consider a grid, so each community is associated to a square on the grid). Once the communities are placed, the nodes start moving according to a few rules. The movement of a node is ruled by a *goal*, that is simply a point on the grid acting as a final destination of movement like in a classical Random-Way Point model. A goal is contained inside a square. The difference with the classical Random-Way Point is that the *goal* is not chosen at random. The first goal of each node is always selected inside the square of its community. Subsequent goals are chosen once the previous goal is reached according to the following mechanism. We already described how a certain number of hosts is associated at time t to a particular square of the grid. Each square, thus, extersts a certain *social attractivity* for a certain host. This represents a measure of the importance of the square in terms of social relationships for the considered host. Formally, given $C_{S_{p,q}}$ (the set of hosts associated to square $S_{p,q}$), it is defined *social attractivity* of that square toward the host i the following value

$$SA_{p,q_i} = \frac{\sum_{j=1, j \in C_{S_{p,q}}}^{n} m_{i,j}}{w}$$

where w is the cardinality of $C_{S_{p,q}}$. The new goal is then randomly chosen inside the square characterised by the highest social attractivity. The model was evaluated by the authors comparing its results to real movement traces, and they showed that the results they obtained are comparable to the behaviour of a real population (for a detailed description of the results, please refer to the original paper).

2.2.2 Purely Synthetic Group Mobility Models

We introduced so far some models where a mobile node moves independently of other nodes; in other words the location, speed and movement direction of the node are not affected by other nodes in the neighborhood. For this reason, these models do not capture many realistic scenarios of mobility. For example, on a freeway, to avoid collisions, the speed of a vehicle cannot exceed that of the vehicle ahead of it. Or, in some applications like disaster relief or battlefield, team collaboration among users exists and the users are likely to follow a team leader. This means that the mobility of a mobile node could be influenced by other neighboring nodes. We did not consider group mobility models for our

experiments until now, but we believe it is important to include the main of them in this work, because an interesting future work could be the evaluation of some protocols when a group mobility is adopted, and to evaluate the different results when different models are adopted.

2.2.2.1 Reference Point Group Mobility Model - RPGM

This model was first proposed in (37) to emulate the movement of a group of soldiers or of a rescue team during disaster relief. In the RPGM model, each group has a center, which is either a logical center or a group leader node; the movement of the group leader determines the mobility behavior of the entire group as follows:

1. The movement of the **Group Leader** at time t can be represented by the motion vector $\overrightarrow{V}^t_{group}$ that defines the motion of the group leader itsels and also the general motion trend of the whole group. Each member of the group deviates from this general vector by some degree. The vector can be either randomly chosen or carefully designed on the basis of certain predefined paths.
2. The movemement of the **Group Members** is significantly affected by that of the leader. For each node, mobility is assigned with a reference point that follows the group movement and upon this predefined reference point, each mobile node could be randomly placed in its neighborhood.

2.2.2.2 Column Mobility Model

This model was first proposed in (38). It represents a set of mobile nodes moving in a certain fixed direction; this mobility model can be used in searching and scanning activity, such as destroying mines by military robots. At the beginning of the simulation, a reference grid forming a column of mobile nodes is defined; each mobile node is then placed in relation to its reference point in the grid and it is allowed to move randomly around its reference point via a mobility model (a Random Waypoint, for example). After some time t, a new reference point RP' will be defined as $RP' = RP + \alpha$ where RP is the old reference point and α is an advance vector determining a predefined offset (the same value for all mobile nodes) used to move the reference grid of the current mobile node. An example of column mobility model is reported if figure 2.9.

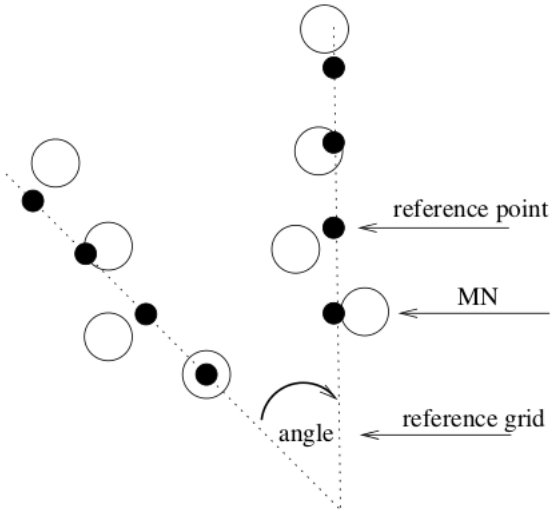


Figure 2.9: An example of movement of three mobile nodes using the Column Mobility Model

2.2.2.3 Nomadic Community Mobility Model

This model represent a slight variation of the Column Mobility Model, and it was introduced in the same work as the previous (38). In this model, each mobile node adopts its own entity mobility model (Random Waypoint for example, or every other model we discussed so far) to roam around a specific reference point; when the reference point is changed, all the mobile nodes move toward it and they start to roam again around the new point. The main difference between the Nomadic Model and the Column Model is that in this case all the nodes share a common reference point instead of having an indivudual reference point placed on a specific column.

2.2.2.4 Pursue Mobility Model

This model too was introduced in (38); as the name suggests this model tries to reproduce a situation in which several nodes follow the movement of a specific node (target tracking is a real example of this movement). The tracked node usually adopts one of the random models introduced so far (most frequently Random Waypoint model), while the follower nodes update their position according to the following equation:

$$new\_pos = old\_pos + acceleration(target - old\_pos) + random\_vector$$

where $acceleration(target - old\_pos)$ is the information on the movement of the tracked node and $random\_vector$ is a random offset different for each mobile node.

2.2.3 Trace-based Mobility Models

We described so far purely synthetic mobility models, i.e. models that are completely described through a set of mathematical rules. The models we present in this section have been realized in an attempt to reproduce real movements in a better way.

2.2.3.1 Weighted Waypoint Mobility Model

This model represents a first step toward the realization of mobility models closer to real behavior of persons. The authors, in fact, observed the problems with purely synthetic models we already remarked, and decided to implement a new mobility model to take into account real persons' choices. They did not collect real traces, but they asked volunteer participants to record their usual behavior for a given period. Once the observation period was over, the authors collected the answers of the participants, and they created a model on the basis of the results of the survey. This way, each different location of the campus was assigned a different weight, depending on its "popularity" (i.e. how many volunteers visited that specific location during the period under investigation). Similarly, the authors defined the transition probability from a given place to another according to the time of the day (e.g. students tended to visit the cafeteria mostly during lunchtime) and a pause duration (how long, in average, a persons stays in a given place). The resulting model, thus, is not exactly based on real traces, but on survey answers; nonetheless it was a very first step in the direction of creating more realistic mobility models.

2.2.3.2 WLAN Mobility Model

The WLAN Mobility Model was first presented in (39). Starting from the observation of the limits of the purely synthetic mobility models we already discussed previously, the authors decided to setup an infrastructure in order to collect real traces on WLAN usage. We are not interested here in the technical details of the whole architecture, for which we refer the intersted reader to the original paper. What is important is to understand the general methodology followed by the authors, clearly depicted in figure 2.10, taken from the original work.

In a few words, the traces recorded from the WLAN access point deployed in the area of interest (a university campus) are processed and used to setup the parameters of the model. Once available, the mobility scenario is used in the

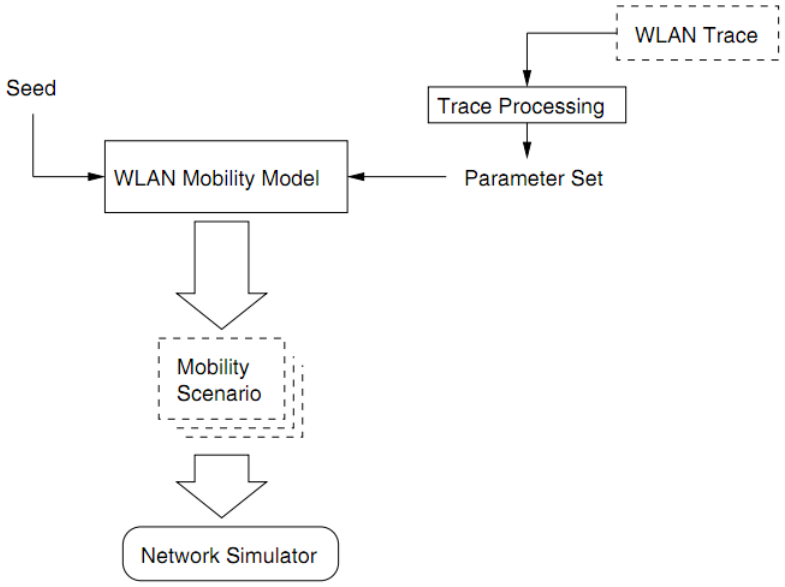


Figure 2.10: The WLAN Mobility Model Architecture

network simulator to evaluate a given algorithm or protocol. Essentially, the idea is that the monitored area is divided in squares, with each square sized on the basis of the coverage of a single WLAN access point. By looking at the collected traces, the authors derive the probability of transitions between adjacent squares; at each iteration, the model thus makes a probabilistic decision to choose the next cell of a given node.

3

Technologies for trace collection

In chapter 2 we focused our attention on synthetic theoretical mobility models, introducing the problem of using theoretical models in terms of poor adherence to realistic movements, in change of the possibility of testing rapidly several different scenarios. We remarked the importance of collecting real movement traces for an appropriate simulation of dynamic mobile social networks, and we explained why using a movement model rather than another has a significant impact on the resulting performance of the tested algorithm. What we missed so far is a discussion on the different technologies currently available to allow the collection of real mobility traces and on the different ways in which the collected information can be represented in a digital form. As pointed out in the introduction, collecting data from a real population of individuals is a difficult and time-consuming task. Setting-up the whole infrastructure for data-collection, programming and distributing suitable devices to a significant number of participants and organizing and analyzing the recorded traces is an hard task. Nonetheless, especially in the last few years, several attempts to collect real data on social interactions were made; despite of the obtained results, we are interested here in reviewing the main adopted solutions and in describing the infrastructure we realized ourselves for our experiments.

The taxonomy we adopt in the following description is to start from the less-accurate technology in terms of social-interaction recording (WiFi), down to the more accurate currently avaialble, to the best of our knowledge (OpenBeacon devices). This chapter will be focused exactly on these 2 aspects: on the first part, we will describe previous experiences on the collection of traces, while on the second part we will focus on the different formats for trace representation, discussing their strenghts and weaknessess.

3.1 Wireless Technologies for Trace Collection

3.1.1 A first step: WiFi

It is important to point out that the works we cite in this section were not intended to record person-to-person interactions, but are currently considered by some authors (e.g. (40)) as pilot-studies to collect data from a large population of individuals.
In the first work, (41), the authors collected for about 4 months statistics on the usage of a campus-wide WLan, using the WiFi networks deployed in the whole campus. They analyzed syslog messages and used the SMNP (Simple Network Management Protocol) to query every 5 minutes each access point (AP) to be aware of how many clients were currently connected to that specific AP. We remark that the information collected in this way gives us a very poor indication on the real social interactions between persons (being connected to the same AP in a given time interval does not imply a social relationship but only, maybe, a physical proximity between 2 individuals). Nonetheless, this was an interesting pilot study on how to collect some kind of information from a population of individuals. Very similar in terms of data collected was the work in (42). Similar in concept, but different on the final aim is the work in (43). The authors collected for about 3 months traces from PDA-users moving inside a university campus using the WiFi network deployed across the same campus. The procedure for data collection is thus similar to the one adopted in the work we discussed previously, but in this case the authors used the traces to refine the standard random-way point mobility model with the help of the real traces, as we deeply discussed in chapter 2. Both studies, although important as a starting point,show the inadequateness of WiFi to record realistic social interactions. The first limitation of WiFi, at least as used in the above-cited studies, is the very low granularity it offers. If only registration of a user to a given access point is considered, without taking into account at least the RSSI of the user's device, it is not possible to state that a person has interacted with another person. It is surely possible to obtain interesting indications on the overall movement of the population (like the most visited place, the average time spent in a given place and so on), but in terms of social interactions we collect very poor information. The second problem is that WiFi does not allow a device-to-device communication as used in the two studies. To record social interactions we need a technology able to record a direct communication or a proximity between 2 users: we thus need a more granular technology, with device-to-device communication capabilities.

3.1.2 A more refined technology: Bluetooth

The next wireless technology we review was used in one of the first medium-scale experiments for the collection of real social interactions (not only mobility traces) in (5) and previously in (40), a technical report written by the same authors on the same topic. The main focus of (5) was on the analysis of information spreading in a delay-tolerant network (the network of contacts, in their case) but we are not interested in the details of this aspect. What is interesting for us is analyzing the infrastructure and the devices they used to record, for the first time (to the best of our knowledge), social interactions between persons. In both works, the authors cite the papers we described in the previous section about WiFi; they conclude that WiFi is not adequate for describing a delay-tolerant-network (their focus was on this kind of networks) since a couple of nodes connected to the same AP may still be not connected to each other, and also because a connection between 2 nodes may happen in a place where no AP is present to record the occurred interaction. We reached similar conclusions for our scenario, as dynamic mobile social networks need the recording of social interactions. For this reason, the authors decided to use the iMote platform, made by Intel Research. According to (40), iMotes (see Figure 3.1) are derived from the Berkeley Mote3 , with the version they used based around the Zeevo TC2001P system-on-a-chip providing an ARM7 processor and Bluetooth support. Along with a 950mAh CR2 battery, each iMote was enclosed in packaging designed to be convenient and easy to carry for test subjects. Two types of packaging were made available : some iMotes were made into keyfobs while others were enclosed in small boxes (see Figure 3.2). Volunteer participants were asked to pick the form factor which allowed them to conveniently keep the iMote with them at all times, with most simply attaching the iMote to their keys.

The authors asked 54 persons among the participants of a conference to carry on the iMote devices during the whole duration of the conference. The iMotes were configured to perform a Bluetooth base-band layer “inquiry”, discovering the MAC addresses of other Bluetooth nodes in range, and the results of inquiry were written to the flash memory. The main limitation of this first experiment, at least for our purposes of recording social interactions, is the use of Bluetooth, as it does not allow a fine-grained recording of social interactions. Also, the small number of correct data recorded (the authors explained that only ≈ 30 sensors recorded useful data) was not sufficient for understanding the social dynamics of a population. In any case, this experiment was an interesting starting point in the right direction, and bluetooth is surely better than WiFi in terms of granularity of contacts. Furthermore, for the first time, interactions between real persons were recorded (even if bluetooth is able to detect a proximity of a few meters, which is still not enough for our purposes), and not simply registration to the same access point, as happened in the case of WiFi.

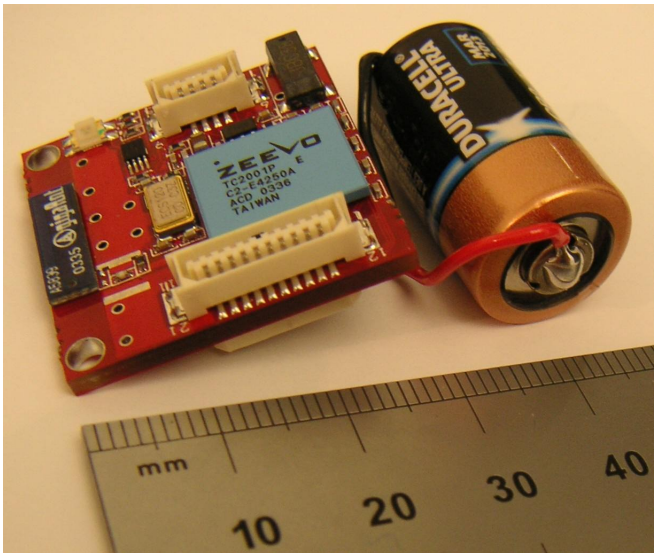


Figure 3.1: An iMote connected to its battery

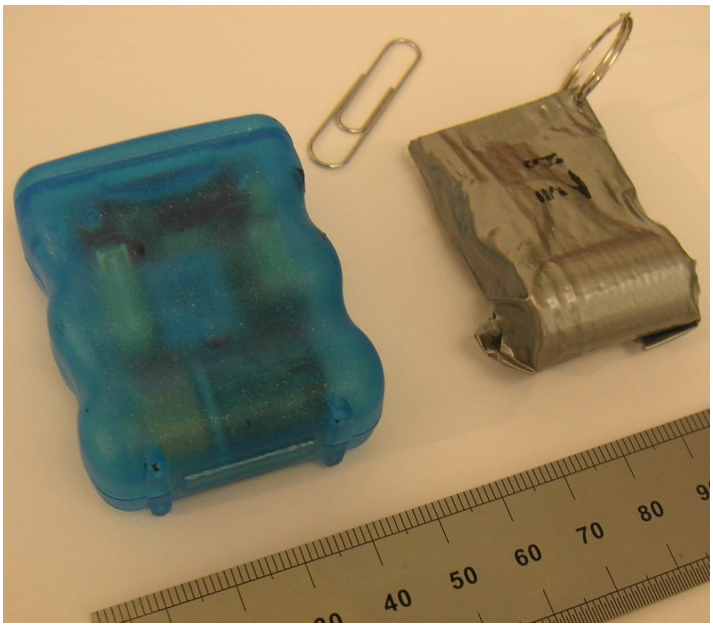


Figure 3.2: Packaged iMotes in boxed form factors

3.1.3 Toward user-centric tracing: Mobile Phones

So far, we presented some technologies that require setting up a whole infrastructure for collecting traces or social interactions. Different in concept, but similar in the fact that some kind of social interaction was collected, was the Reality Mining project (4, 44). It represents the largest mobile phone experiment ever attempted in academia. Its dataset contains thousands hours of continuous data on daily human behavior and contains information on call logs, Bluetooth devices in proximity, cell tower IDs, application usage, phone status. We used this data set for one of our applications, i.e. friendship recommendation (see chapter 4 for details), even if the way social contacts are inferred is a bit different from the one we realized for testing population protocols and distinct counting. Nonetheless, using mobile phones to infer social relationships is a good idea we are thinking about developing as a future work. Simply speaking, the collection was realized by using a dedicated software on the mobile phones (which were, in any case, way less powerful than modern smartphones) able to collect the whole information that may be interesting for better understanding the social behavior of the participants. We found this dataset particularly useful for the friendship recommendation application we will describe in chapter 4, but it still present the same limit as before: it is not sufficiently granular for the purpose of detecting real social interactions.

3.1.4 A finer-grained detection of interactions: Wireless Sensor Networks (WSN)

A wireless sensor network (WSN) is a wireless network consisting of hundreds of spatially distributed autonomous devices (known as sensor nodes, or simply nodes) using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, humidity and so on (45). The development of wireless sensor networks was originally motivated by military applications such as battlefield surveillance. However, wireless sensor networks are actually used in many civilian application areas, including environment and habitat monitoring. In addition to one or more sensors, each node in a sensor network is typically equipped with a radio transceiver or other wireless communication devices, a small microcontroller, an antenna, and an energy source, usually a battery. In a preliminar phase of our work, we decided to use a WSN for collecting traces of social interactions, because it was possible to use the radio transceiver for very low power transmissions, so as to detect a social interaction between users placed at a few meters. This is the reason why in one of the experiments on distinct counting discussed in chapter 4, we implemented our algorithm on real sensor nodes. With the advent and the realization of OpenBeacon tags (described in details in the next subsection), however, we changed our mind since those devices

allow the detection of face-to-face social interactions, detecting a contact between persons when they are ≈ 50cm from each other and facing each other. In any case, WSN represent a first reasonable technology for a fine-grained detection of social interactions. It is sufficient to give each participant a sensor node programmed with a simple application for neighbor discovering, and we have the possibility of detecting and collecting information on the interactions between persons. The main limit of this technology is that, even if more granular than bluetooth (the lowest power level usable for a Telosb sensor node allows for a transmission at about 1.5 meters of distance), it is overwhelmed by OpenBeacon tags, as we will explain in the next subsection.

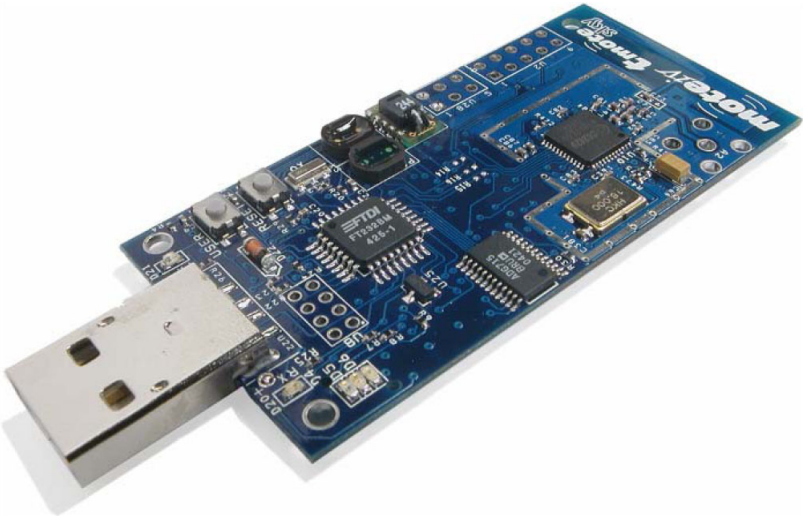


Figure 3.3: A TMoteSky Sensor Node

3.1.5 Face-to-Face Interactions Collection: OpenBeacon tags with Active RFid

The technology that best fits our needs is the active RFid as used in the smart-tags developed by OpenBeacon (46) in range of the SocioPatterns collaboration (3). Since they formed the basis for most of our experiments, we will present here a detailed description of this technology. OpenBeacon produced several different devices, we are mainly interested in 3 of them:

- OpenBeacon Tag (figure 3.4);
- OpenBeacon Tag USB2 (figure 3.5);
- OpenBeacon POE Reader (figure 3.6);

Figure 3.4: OpenBeacon Tag

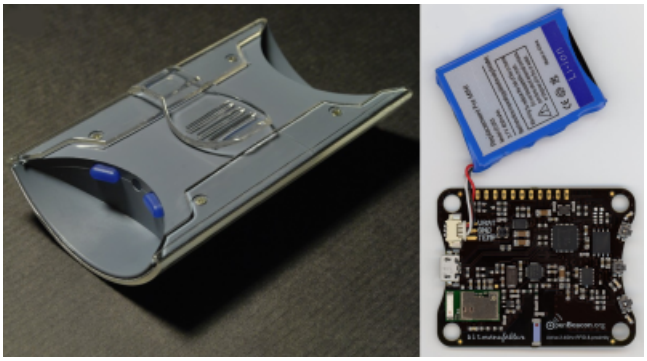


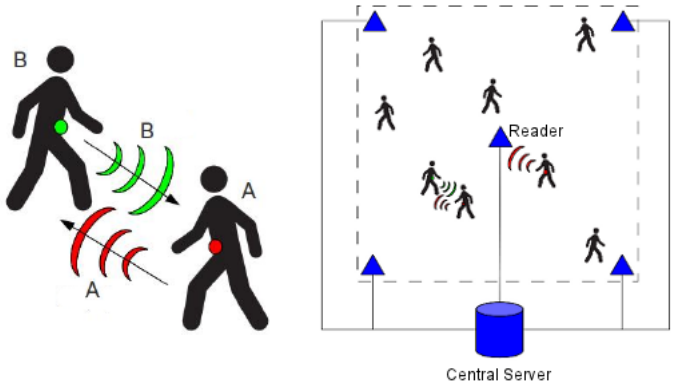
Figure 3.5: OpenBeacon Tag USB 2

Figure 3.6: OpenBeacon POE Reader

From a technical point of view, OpenBeacon Tags are small electronic wearable "badges", equipped with a 2.4 GhZ transceiver made by *Nordic Semiconductor*, and a 14-bit microcontroller made by *Microchip*, the **PIC16F684**. They are powered by a 3-volts li-ion button battery and they can be programmed using an ICSP connection through which they can be connected to a computer for installing the executable program. In the older version (figure 3.4), the battery was replaceable, while in the newest version (figure 3.5), they use a re-chargeable internal battery. Apart from technical differences, the way both versions of tags work is identical, so the description of the protocol they run is unique.
The POE reader (figure 3.6) is a special access point with an ethernet interface and 2 antennas to receive messages from the tags. The ethernet connection is used to send the received messages toward a central collector (server), where data are stored and processed. Essentially, each tag is programmed to periodically transmit a very low power signal containing its unique id; if this signal is detected (or if the tag itself detects a similar signal) by another tag in the proximity, then a contact is recorded, and the id is stored by the other tag. Periodically (once every 2 seconds in our experiments), each tag transmits a report message (called a *Contact* message) to the reader(s), containing the list of the tags encountered in the last time period (see section 3.3 for details). For localization purposes, another type of message is periodically sent by the tags, this time once every 500 milliseconds: a *Sight* message, essentially an "alive" message that can be used to estimate the current position of the tag. The *Sight* messages are transmitted at 3 different power levels cyclically, while the *Contact* message is always sent at the maximum available power level. This behavior is due to the fact that the *Contact* message is the most important message for detecting social relationships, while having 3 power levels helps in the triangulation of the current position of the tag, even if localization is beyond the scope of this thesis.

3.2 Our Data Collection Infrastructure

We reviewed so far the technologies used until now to collect real traces from a real population of individuals. In this section, we will describe in details the methodology we followed to collect real traces ourselves. We had the occasion to collect real traces from 2 different installations in 2 different social contexts, and we describe in the following our approach and a rough evaluation of the traces we collected. The reference architecture for data collection is shown in figure 3.7. Each user wears the active RFID tag shown in figure 3.4. When two active tags are in proximity, i.e. two persons are in proximity, they exchange their id and possibly other useful information[1]. In such way, each tag locally stores the list of recently encountered tags, which is periodically broadcasted to the readers. The readers in turn collect such data and make it available to a central server for off-line processing and traces generation.



(a) When two user meet, they exchange their Id.

(b) Data on tags are periodically broadcasted to readers.

Figure 3.7: The reference architecture for data collection

3.2.1 DIAG Deployment

The first testbed was located in our department and the experiment lasted one week. We collected data from a large area, including several rooms and common spaces, and participants were students that mostly know each other. In this

[1] "A high spatial resolution of less than 1-2 meters is attained by using very low radio power levels for the contact sensing. Furthermore, assuming that the subjects wear the tags on their chest, the body effectively acts as a shield for the sensing signals. This way, contacts are detected only when participants actually face one another. If a sensed contact persists for a few seconds, then given the short range and the face-to-face requirement, it is reasonable to assume that the experiment is able to detect an ongoing social contact (as e.g. a conversation)" (47)

testbed, we used the network infrastructure already available across the whole building, to connect the readers in a dedicated local area network with static IP addresses. Readers were powered by a USB-adapter. Data collected by the readers were delivered to our central server in which they were permanently stored and analysed. The map of this installation is shown in figure 3.8; black dots represent readers. As can be noticed, we roughly placed a reader in each room while some of them were placed in corridors and relax areas. Apart from collecting data on users interactions, this placement of readers gave us valuable information on how and when students use the spaces of the department. This information can allow us to better understand students' behaviour and architect and managers to design more effective spaces and infrastructures.

Figure 3.8: Map of DIAG deployment

3.2.2 MACRO Deployment

The second testbed was during the opening event of a new art exhibition at MACRO (Museum of Contemporary Art in Rome). In this case, only the very large exhibition room was monitored and people were not expected to know each other except for some small groups of visitors. This installation was more complex because no ethernet connection was available in the hall where artworks were placed. We thus decided to set-up a mixed infrastructure using Powerline adapters (D-Link AV 500) for cabled connection over the power line and wireless routers (TP-LINK TL-WR740N) to establish wireless bridges. Also in this case, data were finally collected in our central server. At the same time, a real-time representation of visitors interactions was displayed on a huge screen placed in a dedicated area of the museum; (see section 4.4 for details about the art installation obtained from our experiment). The map of the installation is depicted in figure 3.9. The exhibition's space was divided into 8 "conceptual" areas, and we roughly placed a reader in each of them. This organization potentially allows to relate mobility data to "concepts" and people's interests.

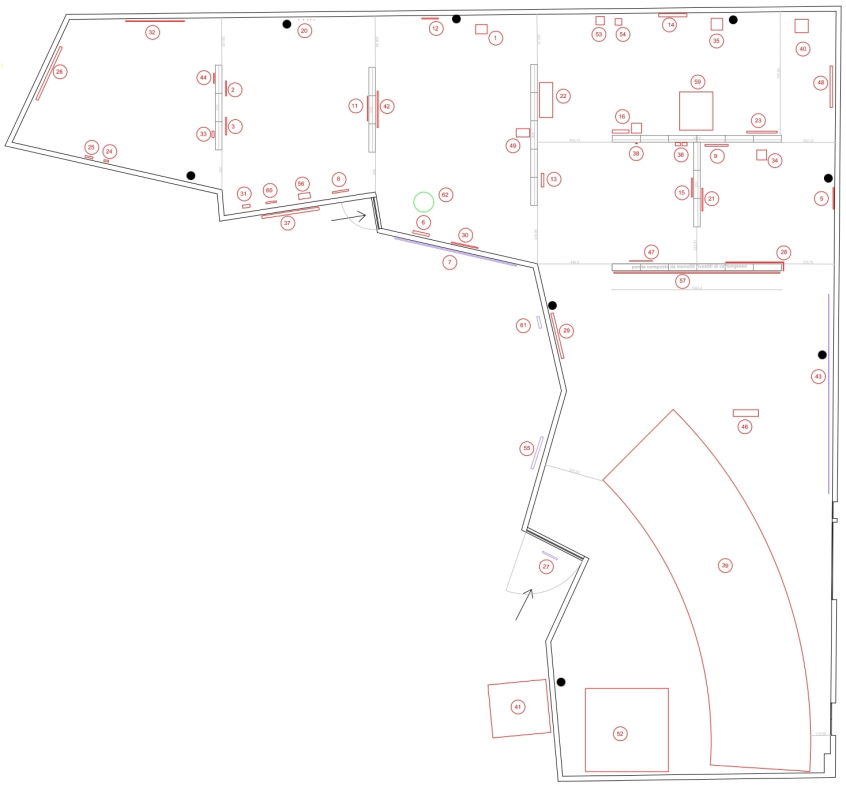


Figure 3.9: Map of MACRO deployment

3.3 Digital Formats for Traces Representation

We explained in the previous section the hard work to be carried on to set-up the whole infrastructure for collecting real traces of social contacts from a real population of individuals. Once the data are collected, however, it is necessary to post-process them in a suitable way in order to properly analyze what has been recorded and eventually to feed the simulator with these real traces. The main problem is due to the fact that, at least for our experiments, data are collected on a text file, with each row representing a message sent from a tag. An example of rows collected is reported below.

Listing 3.1: An extract of the text files we collected

```
C t=1318840850 ip=0xc0a8501c id=1161 seq=0x00004e61
C t=1318840850 ip=0xc0a8501e id=1195 seq=0x00002c61
S t=1318840850 ip=0xc0a85015 id=1191 seq=0x000092c1 strgth=2 last_seen=0
S t=1318840850 ip=0xc0a8501b id=1203 seq=0x0000c2c1 strgth=2 last_seen=0
S t=1318840850 ip=0xc0a8501e id=1153 seq=0x000052c1 strgth=2 last_seen=0
S t=1318840851 ip=0xc0a8501c id=1161 seq=0x00014e81 strgth=0 last_seen=0
S t=1318840851 ip=0xc0a8501b id=1187 seq=0x00008021 strgth=1 last_seen=0
S t=1318840851 ip=0xc0a85015 id=1193 seq=0x00008f41 strgth=2 last_seen=1189
C t=1318840851 ip=0xc0a85015 id=1191 seq=0x000092e1
C t=1318840851 ip=0xc0a8501b id=1203 seq=0x0000c2e1
C t=1318840851 ip=0xc0a8501e id=1153 seq=0x000052e1
C t=1318840851 ip=0xc0a85004 id=1153 seq=0x000052e1
S t=1318840852 ip=0xc0a8501e id=1138 seq=0x000174a1 strgth=1 last_seen=0
S t=1318840852 ip=0xc0a8501c id=1161 seq=0x00014ea1 strgth=1 last_seen=0
S t=1318840852 ip=0xc0a8501b id=1187 seq=0x00008041 strgth=2 last_seen=0
C t=1318840852 ip=0xc0a85015 id=1193 seq=0x00008f61 [1189(1) #7]
C t=1318840852 ip=0xc0a85016 id=1193 seq=0x00008f61 [1189(1) #7]
```

20

The first capital letter indicates the type of message received by the server; a **C** indicates a *Contact Message*; for example, in the last row, [[1189(1)#7] indicates that node 1193 encountered node 1189 in the last time interval. An **S** indicates a *Sight Message*. The field $t =$ indicates the timestamp, added by the central server and expressed as a Unix timestamp, at which the information was sent by the tag. The field $ip =$ indicates the ip address of the reader that collected the information (this information can also be used for a rough localization); the field $id =$ indicates the id of the tag that sent the message, while the fields $strgth =$ and $last\_seen =$ indicates the power at which the message was transmitted (recall from the description of the tags, that they send messages at 4 different power levels) and the id of the last tag encountered. It is obvious that the information collected in this way cannot be immediately used by a simulator or by other tools for the analysis, so we implemented a parser in Java programming language to produce different files in different digital formats, according to our needs. In the subsections below, we describe the formats we decided to adopt and we motivate this decision. We will also present the new DNF format we introduced, explaining the reasons of this choice.

	1001	1002	1003	1004	1005	1006	1007
1001	0	0	0	0	0	0	0
1002	0	0	0	0	0	0	0
1003	0	0	0	0	0	0	0
1004	0	0	0	0	0	0	0.00107
1005	0	0	0	0	0	0	0
1006	0	0	0	0	0	0	0
1007	0	0	0	0.00107	0	0	0

Table 3.1: Weighted Adjacency Matrix Example

3.3.1 Weighted Adjacency Matrix

The first digital format we decided to use for representing the aggregate graph is a weighted adjacency matrix. We already introduced this kind of representation previously, under the name of Interaction Matrix. This is a compact way of representing the interactions between users in terms of the relative strength of each interaction.

Essentially, the first row and the first column contain the ID of all the tags belonging to the network. Column (i, j) contains the number of contacts C between tag i and tag j averaged on the total number of contacts, such that $\sum_{i=0}^{N} \sum_{j=0}^{N} C(i, j) = 1$. The strength of the contacts is thus indicated by the number contained in each column, excluding the first one. This way, we produced a csv (comma separated values) file, and we feed Gephi ((48, 49), described in the next paragraph) with this file and we were able to obtain a first graphical representation of the aggregated file produced by our DIAG experiment (figure 3.10).

The size and the color of a node indicate the number of different contacts it recorded (darker color and bigger size mean more contacts), while the thickness of an edge between two nodes represents the number of contacts recorded between them (taken from the csv file).

3.3.2 GEXF - Graph Exchange XML Format

GEXF is a language for describing complex networks structures, their associated data and dynamics. Started in 2007 at Gephi project (48, 49) by different actors, deeply involved in graph exchange issues, the gexf specifications are mature enough to claim being both extensible and open, and suitable for real specific applications. In this section we want to give the reader a generic introduction to this format, and explain the reason why we decided to use it; the following description is partially taken from GEXF primer, the document explaining the basis of the format (50).
The purpose of a GEXF document is to define a graph representing a network; the GEXF document consists of a gexf element and a variety of subelements: graph, node,

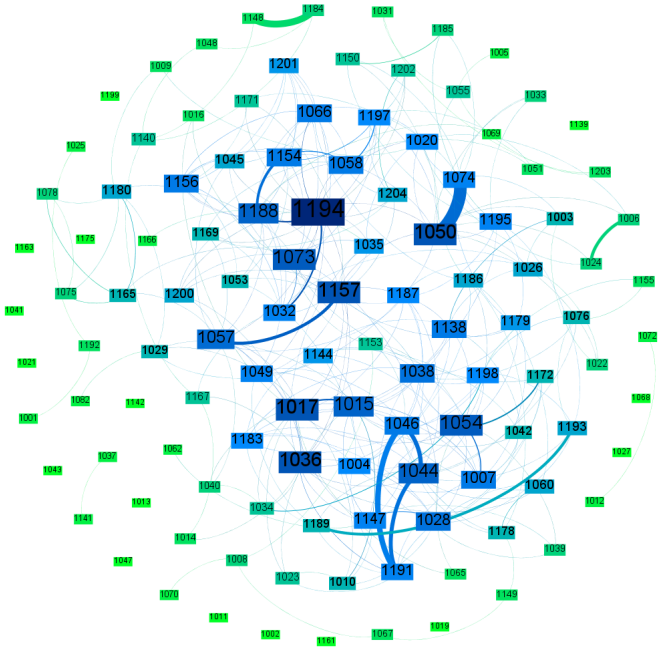


Figure 3.10: An overview of the social network formed during the DIIAG experiment

edge. In the remainder of this section we will discuss these elements in detail and show how they define a graph. First of all, we start giving a very simple example of a graph composed of 2 nodes connected by an edge, reported below.

Listing 3.2: A first example of a GEXF graph

```
<?xml version="1.0" encoding="UTF8"?>
<gexf xmlns="http://www.gexf.net/1.2draft"
 xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xsi:schemaLocation="http://www.gexf.net/1.2draft
http://www.gexf.net/1.2draft/gexf.xsd"
 version="1.2">
<meta lastmodifieddate="20090320">
        <creator>Gephi.org</creator>
        <description>A hello world! file</description>
</meta>
<graph defaultedgetype="directed">
 <nodes>
        <node id="0" label="Hello"/>
        <node id="1" label="Word"/>
</nodes>
<edges>
        <edge id="0" source="0" target="1"/>
</edges>
</graph>
</gexf>
```

The graph generated by the above piece of code is reported in figure 3.11.

The header does not need any additional description, since it is common to all GEXF document and it is a set of XML process instructions. The same is true for the piece

Figure 3.11: The simple example graph from GEXF code

of code defining the *gexf* element. What is interesting for us is spending a few words on the definition of the network topology. The network topology structure containing nodes and edges is called the graph, denoted by a *graph* element. Nested inside a graph element are the declarations of nodes and edges, declared with the *node* element inside a *nodes* element, and an *egde* element inside an *edges* element respectively. Nodes and edges order doesn't matter. Graphs in GEXF are mixed, in other words, they can contain directed and undirected edges at the same time; for our representations, we used undirected graphs. Nodes in the graph are declared by the node element. Each node has an identifier, which must be unique within the entire document. The identifier of a node is defined by the XML-attribute *id*, which is a string. Each node must have a XML-attribute *label*, which is a string.

Listing 3.3: A node declaration

```
<node id="0" label="Hello world" />
```

As for edges, in the graph they are declared by the *edge* element. Each edge must define its two endpoints with the XML-Attributes *source* and *target*. The value of the source or the target, must be the identifier of a node in the same document. The identfifier of an edge is defined by the XML-Attribute *id*. There is no order notion applied to edges. It is possible to specify additional attributes of an edge, like its weight, its being directed or not and so on, but this complete description goes beyond the scope of this section.

Listing 3.4: An edge declaration

```
<edge id="0" source="0" target="1"/>
```

Even if we didn't use this feature during our experiments, in GEXF it is also possible to store some attributes for a node/edge. The concept of attributes in GEXF is the

same as table data or SQL; an attribute has a title/name and a value. Attribute's name/title must be declared for the whole graph. Besides the name of the attribute, a column also contains the type; we will skip details on attributes since we are more interested in describing in more details the representation of dynamic graphs, as the ones we recorded during our experiments. GEXF format includes time support; it is possible to enable it by setting the mode attribute of the graph to *dynamic*. The header below is taken from our DIAG department experiment.

Listing 3.5: Header for a dynamic graph

```
</meta><graph defaultedgetype="undirected" idtype="string" mode="dynamic" timeformat="dateTime">
```

We specified the *dateTime* format to encode time in GEXF. In GEXF it is possible to specify a start time, indicating when an element (node or edge) appears for the first time in the graph, and an end time, indicating when the given element in no longer part of the network. This representation is not enough for our graph, as node and edges may appear and disappear continuously during the whole duration of the experiment. To this end, GEXF covers this case, i.e. if a node or an edge exists only at some timeranges, we use the concept of *spells*. For each node or edge, in other words, we specify the whole list of time intervals (spells) when it is present in the graph. The piece of GEXF file from the DIAG experiment better clarify this concept.

Listing 3.6: Example of nodes and edges with multiple spells

```
<node id="1264" label="1264">
<spells>
        <spell start="2012-06-20T19:55:25.000+0200" end="2012-06-20T19:55:25.000+0200"></spell>
        <spell start="2012-06-20T19:55:27.000+0200" end="2012-06-20T19:55:29.000+0200"></spell>
        <spell start="2012-06-20T19:55:31.000+0200" end="2012-06-20T19:55:37.000+0200"></spell>
        <spell start="2012-06-20T19:55:39.000+0200" end="2012-06-20T19:55:42.000+0200"></spell>
        <spell start="2012-06-20T19:55:44.000+0200" end="2012-06-20T19:55:46.000+0200"></spell>
.
.
.
</spells>
<node id="1262" label="1262">
        <spells>
        <spell start="2012-06-20T20:00:11.000+0200" end="2012-06-20T20:00:11.000+0200"></spell>
        <spell start="2012-06-20T20:00:21.000+0200" end="2012-06-20T20:00:21.000+0200"></spell>
        <spell start="2012-06-20T20:00:23.000+0200" end="2012-06-20T20:00:25.000+0200"></spell>
.
.
.
</spells>
.
.
.
<edge id="118" source="1097" target="1090" type="undirected" weight="1.5663459">
<spells>
<spell start="2012-06-20T22:11:21.000+0200" end="2012-06-20T22:11:21.000+0200"></spell>
<spell start="2012-06-20T22:16:54.000+0200" end="2012-06-20T22:16:54.000+0200"></spell>
.
.
.
</spells>
```

Thanks to the use of spells, we are now able to perfectly represent our graph of social interactions in a graphical view that allows us to perform a very first preliminar analysis of what happened during the experiment. In the figure below you can find an overview of MACRO (figure 3.12 installation respectively

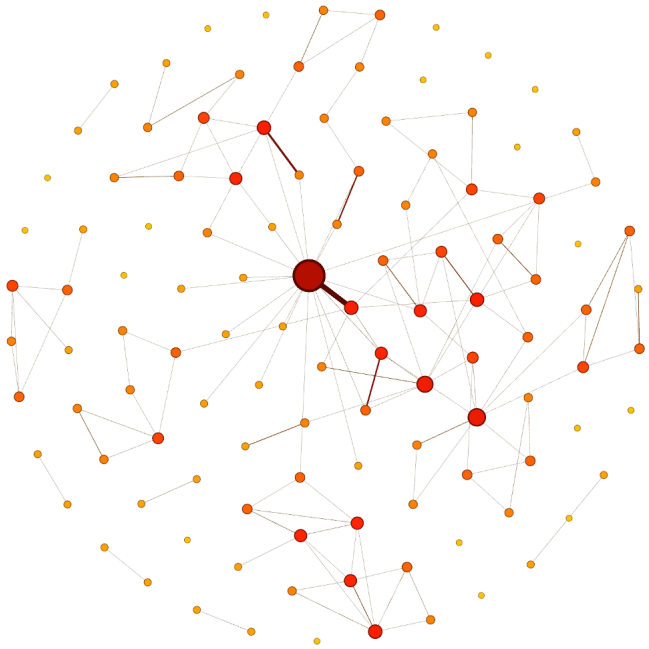


Figure 3.12: An overview of the social network formed during the MACRO experiment

The figure was obtained feeding Gephi platform (49) with the GEXF document we produced parsing the .txt file created by our server during the experiment. The parsing was realized using a Java parser we wrote to this purpose. The graph represent aggregate information on all interactions that took place during the experiments. As in the case of the graph we obtained for DIAG experiment, the size and the color of a node indicate the number of different contacts it recorded (darker color and bigger size mean more contacts), while the thickness of an edge between two nodes represents the number of contacts recorded between them. Once again we stress that the graph is a static, summary representations of the available data-sets, while in our experimental analysis and simulations we used dynamic data. We will come back to this figure in the next chapter, when analysing in details the collected data. For now it is sufficient to say that Gephi was important for us to obtain a fancy graphical representation of our experiments.

3.3.3 JSON - JavaScript Object Notation

According to the official web-site (51) *"JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language"*. In other words, using JSON it is possible to easily represent complex objects with several attributes; it is similar in concepts to GEXF format, but it differs from it because it does not require XML-like labels and the resulting structure is easier to read for humans. In this section we will give a summary description of the file structure we created from the server text file, as there are not so many technical details as for example for GEXF.

Listing 3.7: An extract of the JSON file from MACRO experiment

```
{
  "id": 0,
  "timestamp": "1340214925",
  "visitors": [
    {
      "personid": 6,
      "badgeid": 1264,
      "sex": "F",
      "age": 0,
      "grade": "Primary School",
      "occupation": "Student",
      "nationality": "Italy",
      "guide": false,
      "groupid": 1,
      "readerIDs": [
        "0xc0a85016"
      ]
    },
    {
      "personid": 5,
      "badgeid": 1119,
      "sex": "M",
      "age": 20,
      "grade": "Master",
      "occupation": "Finance",
      "nationality": "Italy",
      "guide": false,
      "groupid": 1,
      "readerIDs": [
        "0xc0a85016"
      ]
    },
    {
      "personid": 4,
      "badgeid": 1135,
      "sex": "F",
      "age": 30,
      "grade": "Master",
      "occupation": "Art",
      "nationality": "Italy",
      "guide": false,
      "groupid": 1,
      "readerIDs": [
        "0xc0a85016"
```

```
        ]
      }
    ],
    "readers": [
      {
        "readerID": "0xc0a85016"
      }
    ],
    "edges": [
      {
        "source_target": [
          6,
          5
        ],
        "power": 0
      },
    ]
}
```

As you can notice from the listed piece of code, we organized information in JSON in timestamps. For every different timestamp, we create a list of the item that were present in that precise instant. Following the hierarchy, in the reported example, we recorded the presence of 3 visitors, each with its own attributes listed in the corresponding curly bracket. The other elements reported in the list above were used to fastly and easily access the list of readers (note that when accessing the list of readers, we lose the correspondence between tag and reader, but we only know how many and what readers received messages from some tags) finally to know how many edges were formed and between what source-target couples. JSON format is easy-to-read and efficient in storing several attributes, but its main drawback is that it uses a lot of memory to store the required information (around 300MB for DIAG experiment).

3.3.4 DNF - Dynamic Network Format

Dynamic Network Format (DNF) is a new file format we created for representing dynamic networks, i.e. graphs that include information about their evolution during time. It's very efficient with respect to other representations because it uses incremental gaps among timestamps. Traditional representations of time-aggregated graph use, for each node or edge, a sequence of timestamps (single or continuous), e.g. GEXF file format use the concept of spells, as we already discussed in the previous subsection. This is obviously not efficient for big networks; just to get an indication, consider that the GEXF file for our DIAG experiment ($\approx$ 5 days) is about 50MB large. For this reason, we decided to think on a different and more efficient solution. First of all, in the listing below we report the header of a .dnf example file. The example is taken from the MACRO experiment.

Listing 3.8: DNF file header from MACRO experiment

```
graphtype:{dynamic}, defaultedgetype:{undirected}
dynamics:{start=1340214925,end=1340223863}
nodeattrs:{badgeid,sex,age,grade,occupation,nationality,guide,groupid}, edgeattrs:{}
```

The header contains indication on the type of graph described, the type of edges, the initial and final timestamps and the attributes of each node and edge. The attribute of a node are then specified when listing all the nodes in the network, as we will show in a successive example; now we focus on the representation of time. The idea behind DNF is very simple: it tries to reduce the file size thanks to gaps among timestamps. For instance, if you have the following dynamic and assume that the global initial timestamp of the experiment is 1335090220:

- node 1001 was present in the following timestamps: 1335090242, 1335090243, 1335090246, 1335090247, 1335090249;

then DNF sets the initial global timestamps to the header of the file and the initial node gap will be the difference between the initial node timestamp and the global initial timestamp (i.e. 22 = 1335090242 - 1335090220). Next gaps will be computed with respect to their previous timestamps, e.g., for the second gap, you have $1 = 1335090243 - 1335090242$, and for the last gap you have $2 = 1335090249 - 1335090247$, so the whole line will be:

```
[1001] (22,1,3,1,2)
```

In case a node/edge is present in successive timestamps, this information is represented in DNF format using the "+" symbol and the number of successive instants (not including the first one); as an example consider the situation below

- node 1003 was present in the following timestamps: 1335090249, 1335090251, 1335090252, 1335090253, 1335090254, 1335090259;

Then the corresponding DNF line will be

```
[1003] (29,2,+3,5)
```

where 29 indicates the initial gap between timestamp 1335090249 and the global initial timestamp, 2 is the gap between timestamp 1335090251 and 1335090249, +3 is a continuous gap for timestamps 1335090251 (not included), 1335090252, 1335090253, 1335090254, and the last gap 5 is the difference between 1335090249 and 1335090254. The reason why the continuous gap is 3 and not 4 is that timestamp 1335090251, the first of the continuous instants, is already represented by gap 2. Including it into the continuous gap, we would not be able to know the dimension of the gap between the first instant of continuous gap (1335090251) and the previous one (1335090249). An identical behaviour is defined for edges, for instance

```
[1010,1089] (39,1,+7,3,+2,10)
```

As for attributes of the nodes, they are written when introducing the new node in the list of nodes. An example of 2 nodes and some edges in reported in the listing below.

Listing 3.9: DNF file header from MACRO experiment

```
[1] {1084,M,50,Bachelor,Other,Italy,false,0} (2221,4,1,4,4,3,1,
4,1,3,1,3,1,3,1,5,3,1,3,1,
3,1,2,+2,2,1,2,2,+2,2,+2,3,
1,3,1,2,2,+5,2,+2,4,2,+2,2,2,
1,4,3,1,3,5,5)

[2] {1152,M,20,Bachelor,Art,Italy,false,0} (7572,20,1,2,+2,9,
2,2,4,29,5,3,1,7,1,4,1,2,+2,4,4,
13,3,1,3,1,4,1,4,21,7,9,1,11,1,4,
1,11,1,4,1,4,8,16,9,9,2,2,4,4,5,
3,1,3,1,4,17,4,5,4,4,2,2,9,7,1,4,4,
4,1,3,1,4,4,4,9,4,4,2,3,2,2,2,+2,2,
+2,2,+2,5,4,4,46,5,8,4,13,4,9,12,3,
2,3,1,3,1,13,7,1,2,+2,3,1,5,3,+3,2,
3,5,4,1,3)
.
.
.
[1,2] (148,110,64,79,47,132,16,3,2,29,3,1,26,29,5,192,1544,20,5,4,4,39,
155,211,1215,708,1,3,1,3,1)
```

Each new node is added to the file preceeded by square brackets and an identifier (not the ID of the tag); inside the curly brackets, all the attributes of the current tag are reported (the information specified in the header of the .dnf file), and then the list of timestamps and gaps as we discussed before. The same is true for edges: in the example reported above, we reported the intervals in which the edge between tags 1 and 2 exists. Please note that also in this case we are not referring to the ID of the tag itself, but to the new identifier we assigned to each entry inside the file. So, the edges are between tags whose real experiment ID is 1084 and 1152. Thanks to the optimization of DNF format, we were able to reduce the file size from $\approx$ 50MB to $\approx$ 2MB; it is quite evident how this new representation could be extremely useful for large sets and/or very long experiments.

3.4 Excursus: on the collection of large real mobility traces and the reliability of network simulators

Until now, we remarked many times how the use of mobility traces collected from a real population of persons is fundamental to evaluate an algorithm/protocol in a realistic setting. We pointed out that synthetic traces, although useful, accurate, and highly flexible (in the sense that they allow the developer to test its solution in several different scenarios by fastly modifying simple parameters and repeating the simulation) have the limit of being quite far from reality. But an obvious question may arise at this point: what if, after a long and tedious work for setting up a whole infrastructure to collect real traces, the simulator we decide to use is not able to produce reliable results? Apart from the theoretical mobility model, a simulator is responsible of reproducing the behavior of several other elements of real experiments: radio devices, signal propagation, simulation field, and so on. What if the simulator fails in reliably reproducing one or more of this fundamental elements? Answering to this question is of primary importance, and this

is the reason why we decided to accurately evaluate 2 of the most diffused network simulators.

3.4.1 Simulations and reliability of simulators

The difficulty of deploying and managing large-scale testbeds is the main reason why only a few datasets of mobile networks are currently available. More generally, this is also true if we consider static ad-hoc networks, like for example Wireless Sensor Networks deployed in buildings or in galleries for structural monitoring, or in outdoor areas of specifical archaeological interest where some environmental parameters (humidity, temperature) must be continuously controlled: currently, only a few real permanent testbeds have been deployed or used by researchers to validate their proposed algorithms and protocols; this validation is usually presented only through the use of network simulators. Testbeds and simulators are two important and complementary design and validation tools for the development of networking solutions. An ideal development process should start from the theoretical analysis of the proposed solution providing bounds and indication of its performance, verified and refined by simulations and finally confirmed by testbeds. Even if it is well-known that simulation results only approximate real ones, and even if simulators are frequently the only means through which a new algorithm is tested, only very few works, such as (52), (53), (54), (55) have investigated the reliability of simulators, comparing simulation results to testbed ones on the same networking scenario. Nowadays the vast majority of experimental works on computer networks (including the testing and evaluation of new algorithms) is still based on simulation results only, and the main reasons for this choice can be summarized in the following paragraphs.

Ease of Use Many of the logistical challenges related to the development, deployment and debugging of realistic large-scale wireless networks have gone unmet. Manually reprogramming nodes, deploying them into the physical environment, and instrumenting them for data gathering or communication is difficult and time-consuming. Furthermore, wireless networks face many problems that do not arise so acutely in other types of networks. First of all they are strongly power-constrained: sensor nodes (e.g. OpenBeacon tags or TmoteSky nodes) are battery-powered and battery replacement is either uneconomic or unfeasible in most of the envisaged scenarios. The tiny sensor nodes are made of low-cost hardware and are thus fragile and prone to failures. Moreover, these devices can be deployed and operate in hostile environments, and consequently unexpected node failures is a likely event. Finally, node programming is prone to bugs that typically arise in distributed, embedded and wireless systems. In most cases, bugs are hard to detect because the limited communication and computational resources prevent nodes from freely storing and transmitting debugging information, as this quickly depletes energy and reduces network lifetime. As a result, once a wireless network

is deployed, visibility into the network drops dramatically. Although there is an effort in designing efficient debugging tools there is still a dearth of them (56). Due to the complexity and difficulty of implementing real testbeds, simulators are widely used. Simulations allow researchers to validate wireless networks solutions before deployment, so as to reduce the need for corrective actions once the network is actually operating, and furthermore they enable the large scale experimentation of protocols and applications in a flexible and scalable environment.

Scale Factor Wireless networks (Wireless Sensor Networks, social networks and so on) are usually conceived for very large and dense deployments. As an example of WSN, consider an environmental monitoring scenario and for the sake of simplicity assume a symplistic grid deployment with radio range of 100 meters, which is an upper bound for IEEE 802.15.4 radios range in outdoor environments. Under these assumptions, a node placed in the center of a square, covers a surface of an hectare. To have a rough idea of the number of nodes required to monitor a national park, consider that the Pollino National Park, the largest Protected Area among the most recent parks of Italy, extends 182 thousands hectares and Vesuvio National Park, which contains the most important active volcanic group of continental Europe, is about 8.5 thousands hectares. Even if a WSN covering the whole surface of a national park would be probably unrealistic, a random deployment in a three-dimensional environment (most of those parks are in mountain areas) will substantially increase the number of required nodes per hectare. The same is true if you consider a deployment of a wireless social network experiment inside, for example, a mall. It is realistic to think that a big experiment could involve several thousands of persons moving randomly inside the whole monitored area. In any case, current testbeds (we refere here to static WSN testbeds, as we are not aware of existing social networks testbeds publicly available, not considering datasets of past experiments) can be made of few (one or two) thousands nodes at most (57, 58), but they are typically made of only few hundreds (59, 60, 61, 62) while simulators can manage up to several hundred thousands nodes (63, 64).

Reproducibility of Results "Reproducibility is a core principle of science. For computational experiments to become reproducible, one needs to develop a system for linking scientific publications with computational recipe" (65). Simulation tools, though simplistic, are controlled environments where reproducibility can be easily achieved. Consequently, simulation results can be verified by any researcher around the world and can be used as a baseline to evaluate the performance of new solutions, supporting an incremental process of knowledge accumulation and dissemination. However, reproducibility not always receives the attention it deserves. As far as we know only one workshop (66) has explicitly explored what is needed for network research to ensure widely reproducible results.

Until now, we did not focus our attention on another key aspect: the importance of using (and setting correctly) one network simulator instead of another; it is obvious that using a simulator that perfectly reproduces the movement of humans, but completely neglects the effects of, for example, distance on wireless communications is completely unuseful if we want to evaluate the behavior of a routing protocol or the efficiency of a gossip algorithm for example. On the other hand, a very accurate channel modeling and simulation of the physical reality is not needed when for example we just want to evaluate the convergence property of a given algorithm. When validating a new algorithm or protocol by means of simulations, it is extremely important to have indications on the validity of the recorded results, and to know how close they are to reality; thus, it is crucial that the results obtained from simulations are close to the ones expected from the real deployment. We will describe in the following the main characteristics of several simulators we analyzed and tested, and we will explain why, for some experiments, using a custom simulator (i.e. a program written from scratch to simulate a specific experiment) is a necessary choice.

NS-2 Simulator NS-2 is probably the most widely adopted network simulator (67, 68), intended to simulate several different types of networks. NS-2 is an object oriented simulator, written in C++, with an OTcl interpreter as a frontend (69). The simulator supports a class hierarchy in C++ (also called the compiled hierarchy), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy). The two hierarchies are closely related to each other; from the user's perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter and are closely mirrored by a corresponding object in the compiled hierarchy. NS-2 uses two languages because the simulator has two different things it need to do. On the one hand, detailed simulations of protocols require a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters of configurations, or quickly exploring a number of scenarios. In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. Then in the OTcl script provided by the user, we can define a particular network topology, the specific protocols and application that we wish to simulate (whose behaviour is already defined in the compiled hierarchy) and the form of the output that we wish to obtain from the simulator. NS-2 is a discrete event simulator, where the advance of time depends on the timing of events which are maintained by a scheduler. An event is an object in the C++ hierarchy with an unique ID, a scheduled time and the pointer to an object that handles the event. The scheduler

keeps an ordered data structure (by default NS-2 uses a simple linked-list) with the events to be executed and fires them one by one, invoking the handler of the event.

Castalia Framework for OMNeT++ Castalia is a C++ simulator designed specifically to simulate Wireless Sensor Networks. It is based on the OMNeT++ platform and it can be used by researchers and developers who want to test their distributed algorithms and/or protocols in a realistic wireless channel and radio model with a realistic node behaviour especially relating the access of the radio (70). The modularity, reliability, and speed of Castalia is partly enabled by OMNeT++, an excellent event-driven simulator (71). OMNeT's basic concepts are modules and messages. A simple module is the basic unit of execution. It accepts messages from other modules or itself and according to the message it executes a piece of code. The code can keep state that is altered and can send new messages (or schedule new messages to be sent). When a node has a packet to send this goes to the wireless channel which then decides which nodes should receive the packet. The nodes are also linked through the physical processes that they monitor. There can be multiple physical processes, representing the multiple sensing devices that a node has, as well as multiple wireless channels to represent the multiple radios that a node might have.

Shawn Simulator Shawn (64) is a recent simulator that differs from the simulators disussed above because it follows a different approach, aiming at simulating the effect caused by a phenomenon and not the phenomenon itself. The following description is partially taken from the official web page of Shawn (72)

Shawn differs in various ways from the above-mentioned simulation tools, while the most notable difference is its focus. It does not compete with these simulators in the area of network stack simulation. Instead, Shawn emerged from an algorithmic background. Its primary design goals are:

- Simulate the effect caused by a phenomenon, not the phenomenon itself.
- Scalability and support for extremely large networks.
- Free choice of the implementation model.
- Direct portability of the source code on iSense sensor devices

As for the first bullet, this has several implications on the simulations performed with Shawn. On the one hand, they are more predictable and there is a performance gain since such a model can be implemented very efficiently. On the other hand, this means that Shawn is unable to provide the same detail level that other simulators provide with regard to physical layer or packet level phenomena. However, if the model is chosen well, the effects for the application are virtually the same. Imagine two implementations of a MAC layer: one abstract implementation that yields an increased packet loss on high local traffic and one that calculates interference for single packets using radio propagation models. Both will produce similar effects on the application layer. It must be mentioned though that the interpretation of obtained results must

take the properties of the individual models into account. The second and third bullets are interesting characteristics, but they are not unique in Shawn simulator. What is extremely interesting and different from other existing solution is the immediate portability of the source code. Thanks to a dedicated library, in fact, the Shawn simulator library and the iSense application are linked together and form a binary that is executable both on the simulator and on the real devices. This is the reason why, in a first time, we selected Shawn for our experiments on the distinct counting, as we will explain later.

Netlogo Simulator NetLogo (73) is a programmable modeling environment for simulating natural and social phenomena. It was authored by Uri Wilensky in 1999 and has been in continuous development ever since at the Center for Connected Learning and Computer-Based Modeling. NetLogo is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of "agents" all operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from their interaction. One important thing to remark is that NetLogo does not take into account low-level effects of message propagation, radio modeling, collisions and so on. It simply models the evolution of a specific system over time, giving indications on the behavior of the system itself. This is the main reason why we decided to adopt this simulator for the simulation of our population protocols on the social networks, as in the first part of our work we were more interested in the convergence of the protocols themselves in "ideal" conditions rather than considering real world effects. Effects that we decided to consider as long as simulated results were encouraging in this sense.

Custom Simulators Sometimes, existing simulators are not suited to simulate exactly the event one is interested in analyzing. In these cases, it is necessary to write a custom simulator from scratch to model the phenomenon of interest. Using a custom simulator is usually not a good choice, since existing simulators have been used for many years, programmed and tested by thousands of researchers all around the world, while a custom simulator is implemented and used by a single research group, and it is more exposed to errors, bugs and similar problems. Nonetheless, in some cases, an appropriate simulator does not exist, and it is thus necessary to implement a dedicated solution. It was the case of our work on collaborative filtering (see chapter 4). We were interested in recommending a friendship between two users (evaluated in terms of link prediction, as we will describe) pn the basis of the similarity of 2 user profiles. In that case, it was easier for us to write a dedicated solution, since we were not aware of existing simulators doing precisely this work.

As we already pointed out when introducing technologies, in the initial phase of our research activity on dynamic mobile social networks, we focused our attention on

Wireless Sensor Networks for reproducing, analyzing and collecting traces from a real population, since the OpenBeacon technology was not yet available at the time. For this reason, we decided to evaluate the accuracy of 2 of the above-mentioned simulators for wireless sensor networks in order to understand which one was the more reliable and suitable to our need. The approach we followed was a comparison between some metrics recorded on a real indoor testbed (Motelab permanent testbed (61), deployed at Harvard University) of several Telosb sensor nodes and those recorded on the simulators when modeling exactly the same network topology. In the next section we present and discuss our findings (74). The simulators we analyzed are NS-2 (69) and Castalia. (75).

3.4.1.1 Castalia and NS-2 as Simulation Tools for WSNs

A critical job in WSN simulators is the modeling of sensor nodes and in particular the strict correlation, typical of embedded systems, that exists between hardware and software components. In figure 3.13 the general architecture of a real WSN node is depicted while figure 3.14 shows how the two simulators model a sensor node. As clearly emerges from the figures, Castalia is based on a modular architecture that well reflects the general structure of a wireless sensor node, while NS-2 provides a less modular architecture.

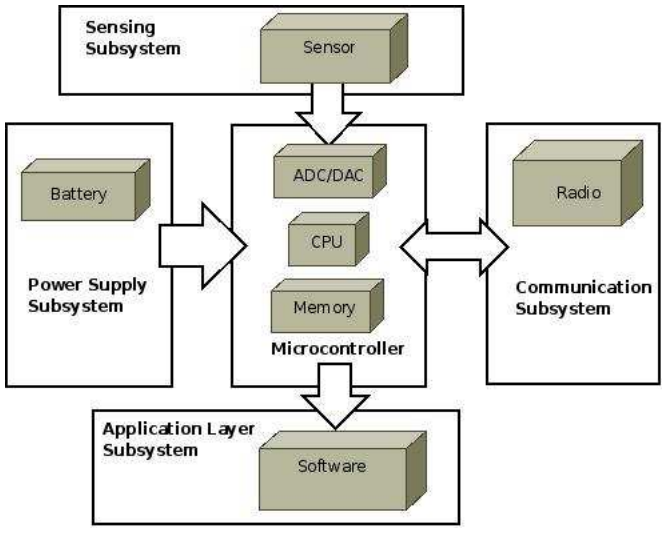


Figure 3.13: General architecture of a wireless sensor node

In the following we go further in our analysis, extending the comparison between Castalia and NS-2 to some of the main components involved in our experiments.

Physical Layer - Wireless Channel To perform our simulations we used the *log-normal shadowing path loss model* described in (76) and characterized by the following

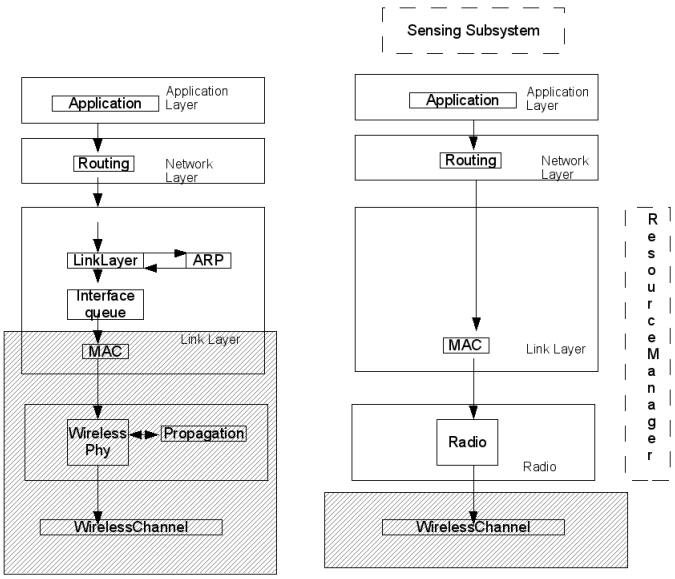


Figure 3.14: NS-2 node model on the left and Castalia node model on the right

formula:

$$PL(d) = PL(d_0) + 10\eta \log_{10}(\frac{d}{d_0}) + \chi_\sigma \tag{3.1}$$

where d is the transmitter-receiver distance, d_0 is a reference distance, η is the path loss exponent and χ_σ is a zero-mean Gaussian random variable (in dB), that in the most general case is a random process function of the time, with standard deviation σ (shadowing effects). The received signal strength at a distance d is the output power of the transmitter minus $PL(d)$. We will discuss later how a simple tuning of the parameters of (3.1), can greatly improve the accuracy of simulation results.

Interference Models In NS-2, irrespective from the number and characteristics of possible interfering transmissions, only one affects the Signal Interference Ratio (SIR) at the receiver. This implies that the SIR obtained when just one node is interfering can be by chance the same as when multiple nodes are interfering. In Castalia, 3 interference models are provided:

- *No Interference* - in this case interferences are not considered even if several nodes are transmitting concurrently. It's like the nodes are transmitting on different frequencies.
- *Simple Collision Model* - if 2 transmissions partially overlap, a collision is assumed and packets are both discarded.

- *Additive Interference Model* - the SIR is calculated considering all the possible interferences (rather than just one as in NS-2)

Physical Layer - Radio and Link Layer - CSMA/CA In Castalia, the *Radio* module models most of the main features of a generic low power modern communication hardware: it supports carrier sense and multiple states with different power consumptions and delays for each state transition (e.g. from idle to TX). The default radio parameters are set according to the datasheet of the CC2420 (77) radio chips which is the radio of the Tmote SKY nodes (78) deployed in Motelab. In NS-2, the radio hardware component is not explicitly modeled by a single C++ class; instead, radio functionalities are performed by the *WirelessPhy* and the *Propagation* classes. The MAC on testbed nodes is a simple CSMA/CA protocol (i.e. a carrier sense collision avoidance MAC protocol) that has been implemented in Castalia customizing the built-in MAC_JustCarrierSense protocol, while in NS-2 we wrote from scratch a suitable C++ class. We stress again that in NS-2 there is not a clear distinction between MAC functionalities and channel functionalities.

3.4.1.2 Description of Experiments

In our approach, we used the results of some experiments on the third floor of the Motelab permanent testbed as a baseline to evaluate the accuracy of simulation results. We first reproduced in the simulators (i.e. Castalia and NS-2) the same geometric configuration of the testbed (i.e. node placement). Then we run some simple protocols both on the testbed and on the simulators and we collect metrics of relevance. Finally we compare the metrics of the simulators to those of the testbed to evaluate the accuracy of the simulation results. The considered testbed is made of 34 Tmote SKY nodes (78), running Tinyos (79), deployed in an indoor area of approximately 81x30 meters (see figure 3.15).

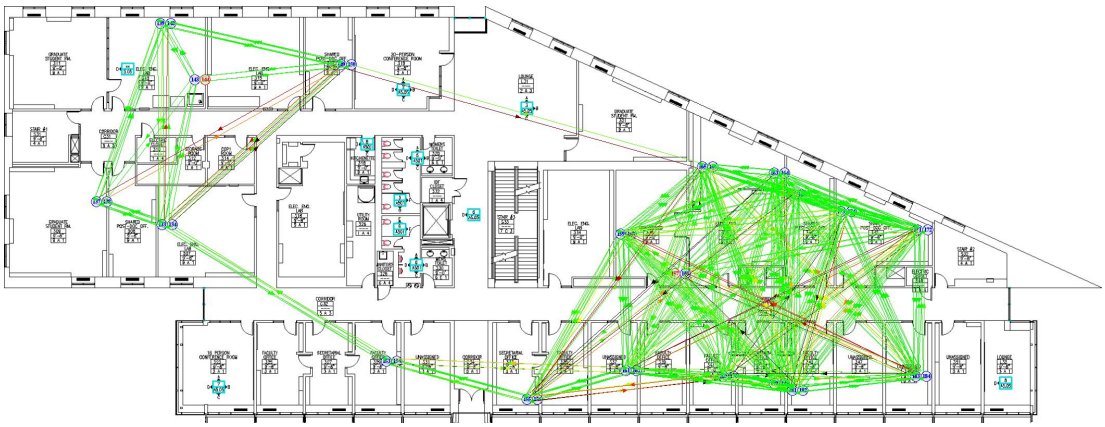


Figure 3.15: A map of the third floor at Motelab

Network-setup

1: $setn_{id} = 0$ Number of messages received from node id
2: $setN = \varnothing$ Neighborood
3: $sett$ Number of messages to send
4: $setT \leq t$ Threshold on the number of received messages
5: $setHC = NumNodes$ Local hop-count
6:
7: **if** sink **then**
8: $HC = 0$
9: **end if**
10: **for** $i = 0$ **to** $NumNodes$ **do**
11: **for all** nodes with $HC = i$ **do**
12: bcast message($HC + 1$,id) t times
13: **end for**
14: **end for**
15: **receive message(**Rec_HC**,id):**
16: **if** $n_{id} \geq T$ **then**
17: $insertidinN$ id is in the neighborhood of the current node
18: **if** $Rec\_HC < HC$ **then**
19: $HC = Rec\_HC$
20: **end if**
21: **end if**

Figure 3.16: Network Set-up

We consider a simple experimental set-up in order to have a clear and manageable environment in which the behavior of the network is not affected by complex technical details. The rationale behind this choice is that, if a distance between simulation and testbed emerges in a very simple and clear environment, this distance would be further extended on more complex ones. The reference protocols on which we collect the metrics are: a *Network Set-up* (see protocol 3.16) and *Gossip* protocols.

At the end of the Network Set-up protocol, each node s has an *hop count* HC^s which represents the distance to the sink (the lower is the closer) and a set N^s which contains its neighbors. Once the Network Set-up protocol has completed, the Gossip protocol works as follows: a node s receiving a message from node j with $HC^s < HC^j$ forwards it, otherwise the message is discarded. It is worth noting that, even if the considered protocols are particularly simple, they can be considered as primitives to construct more complex protocols. As an example, the set of neighbors is exploited in most of the data delivery protocols and the hop count is the basic ingredient for the

construction of a broadcast tree. Furthermore, albeit simple, the gossip protocol allow us to load the network with a significant amount of traffic so as to verify its performance in a stressing configuration. The Gossip protocol is used to propagate data toward the sink from a specific region of the network. We considered three generators placed in the top-left side of the map in figure 3.15 (the sink is bottom-right), generating data at two data rates: i) 36 byte packet every 60 milliseconds, corresponding to a data rate of about 5kbps; according to (80) a tri-axial accelerometer generating 16 bit samples at 100Hz, ii) one 36 byte packet every 6 milliseconds, corresponding to about 50kbps which according to (81) is the data rate to transmit a (120 x 125) pixels image with 16-gray levels[1]. We evaluated the performance of the simulators in two different settings:

- *Default*, namely simulators are used as black-boxes where users do not tune any of the physical layer parameters and default values are used. In this case the default values of the expression (3.1) are $\eta = 2.4$, $PLd0 = 55$, $d0 = 1$, $\sigma = 4$ and all links are symmetric.
- *Tuned*, where only the path loss exponent η is tuned in the range $\{3.0..3.5\}$ according to the observations made in (83) regarding the simulation of indoor environments.

3.4.1.3 Metrics and Results

We first evaluate the accuracy of the simulators in terms of *Connectivity Accuracy*; namely for each node s, we evaluate the similarity between its neighborhood in the testbed N^s_{TE} and in the simulators N^s_{SI}.

To calculate this value, we adopted the Jaccard coefficient, which is a standard measure for analyzing the similarity between sets and it is defined as follows:

$$J(N^s_{TE}, N^s_{SI}) = \frac{|N^s_{TE} \cap N^s_{SI}|}{|N^s_{TE} \cup N^s_{SI}|}.$$

Observe that when $J(N^s_{TE}, N^s_{SI}) = 1$ the two sets completely overlap, while if $J(N^s_{TE}, N^s_{SI}) = 0$ the sets are disjoint. We then evaluate the *Topology Accuracy*. Our protocols do not actually define a "conventional" routing topology (e.g. a routing graph), but each node s is characterized by the hop count HC^s which represents its distance to the sink. Hence, here we simply compare the number of nodes with a given distance to the sink (i.e. hop count) in the simulators and in the testbed. Finally we evaluate the *Packet Delivery Ratio*, namely the percentage of packets correctly received by the sink over the total number of packets generated by generators and propagated toward the sink through the Gossip protocol. All the considered metrics are averaged on 10 independent runs.

[1] 50Kbps is the maximum achievable data rate in tmote sensor nodes equipped with the CC2420 radio transceiver (82)

Connectivity Accuracy In this section we evaluate the accuracy of the simulators in terms of connectivity. Instead of considering the quality of the link connecting two nodes, we consider an indirect and aggregate metric: the similarity of the sets of neighbors in the testbed and in the simulators. More formally, denoted by n the number of nodes, and N^s_{MO}, N^s_{NS} and N^s_{CA} the neighborhood of node s in motelab, NS2 and Castalia respectively, we evaluate the average similarity of these sets by means of the Jaccard coefficient, as follows:

$\frac{1}{n}\sum_s J(N^s_{MO}, N^s_{SI})$ where $SI = \{NS, CA\}$.

It is worth nothing that connectivity accuracy is probably the most fundamental property simulators should guarantee, because the performance of all the subsequent phases of any network protocols, such as interest dissemination, data collection etc. clearly depends on the reliability of connectivity. The elements of a neighbor set can be extremely dynamic also in a static topology like the one we are considering. To take into account this issue, the set N^s of neighbors of s is defined as follows:

A node $j \in N^s$, if and only if it receives at least T of the t messages broadcasted by s in the *Network Set-up* protocol 3.16. The parameter T is a *Link Threshold* and it is used to guarantee more reliable and stable links. In the rest of the paper we will consider the following cases:

- **[Ne1]** $T = 1$, j has to receive at least one message from i;
- **[Ne2]** $T = 0.5t$, j has to receive half of the messages;
- **[Ne3])** $T = t$, j has to receive all the messages.

In our experiments, we considered $t = 10$ and $t = 100$ and for both cases we obtained similar results. For this reason in the following we only discuss the $t = 10$ case.

In figure 3.17 we show the behavior of the average Jaccard coefficient between the neighbors' set of the testbed and of the simulators, when the path loss exponent η in formula 3.1 ranges from its default value (i.e. 2.4) to 3.5. This is the only simple tuning that we will perform. We stress that, this operation can also be done by unskilled users with a minimal effort and as we will see in the rest of the paper can be greatly beneficial in improving simulation accuracy. A first remarkable result is shown in the upper figure in 3.17. In this case NS-2 produces a complete connectivity graph (i.e. each node is in visibility of all the others) for all the values of η, which is significantly different from the actual graph produced by motelab. This behavior is only mitigated with the other definitions of neighborhood (see second and third figures in 3.17), where the effects of packet drops due to errors or collisions are more relevant. Castalia always shows better performance in all the three settings, in particular when $\eta = 3.0$. It is worth noting that when η is set to the default value 2.4, we obtain the worst results. This observation confirms that, in indoor environment as in this case, the path loss exponent has to be carefully tuned (83) and consequently the use of simulators in their *Default* configuration may produce unreliable results. Recalling that an higher Jaccard

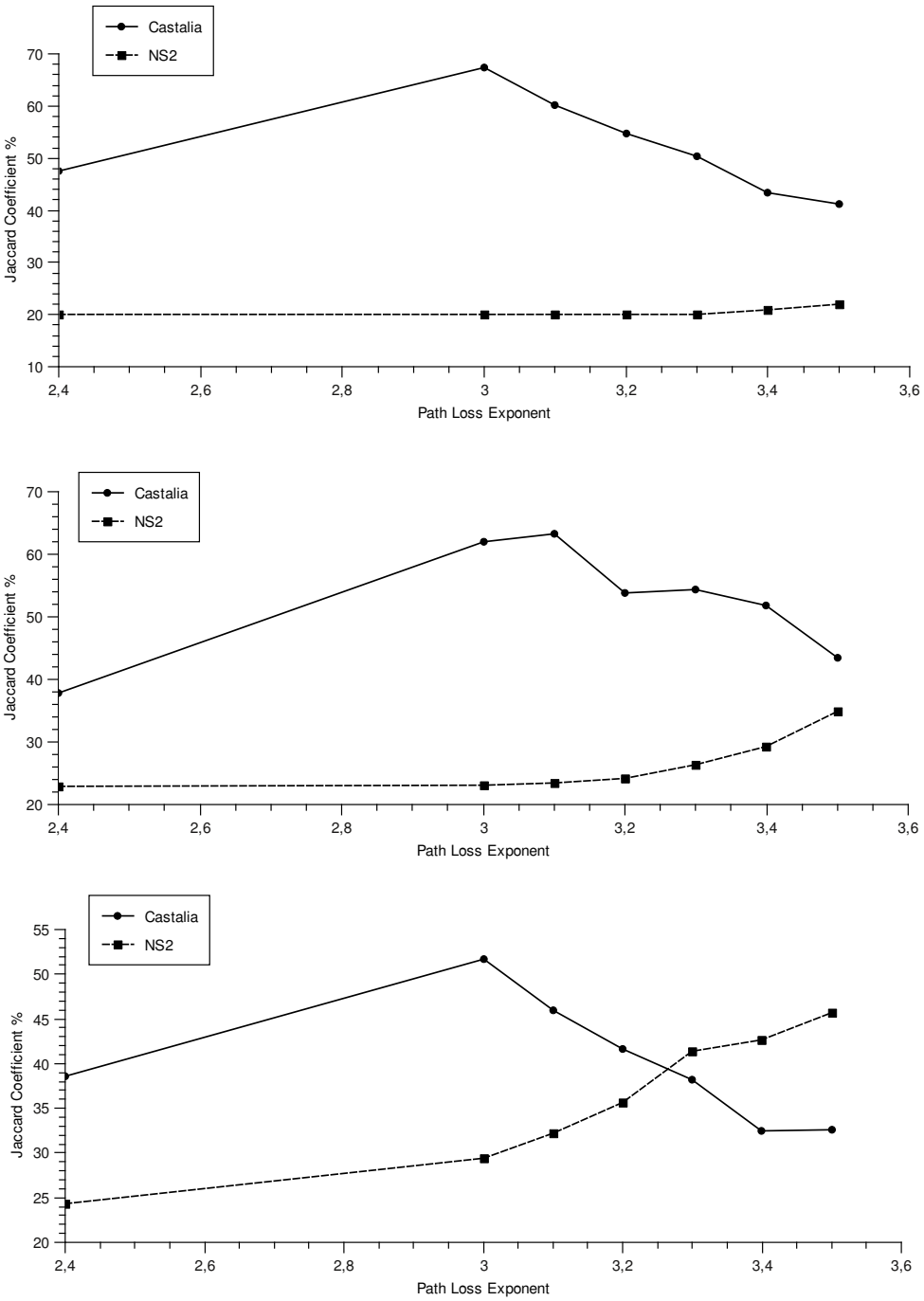


Figure 3.17: Connectivity Accuracy: Jaccard coefficient as a function of η

coefficient indicates an higher similarity and thus an higher connectivity accuracy, the value of $\eta^*_{NS} = 3.5$ (i.e. the value providing the best accuracy results in NS-2) while

the value of $\eta^*_{CA} = 3.0$. In the rest of the paper we will use these parameters in the *Tuned* configuration to evaluate the other metrics of interest.

Topology Accuracy The receiver sensitivity RS is the the lowest power level at which the receiver can detect an RF signal and demodulate data. In other words, signals with received power below the RS are not intelligible.
Tmote SKY nodes are equipped with the CC2420 radio that has a receiver sensitivity of -95DBm (78). When $RS = -95DBm$ and the [Ne1] and [Ne2] cases are considered, Castalia always produces good results; in particular when $\eta = \eta^*_{CA}$ (see figure 3.18).

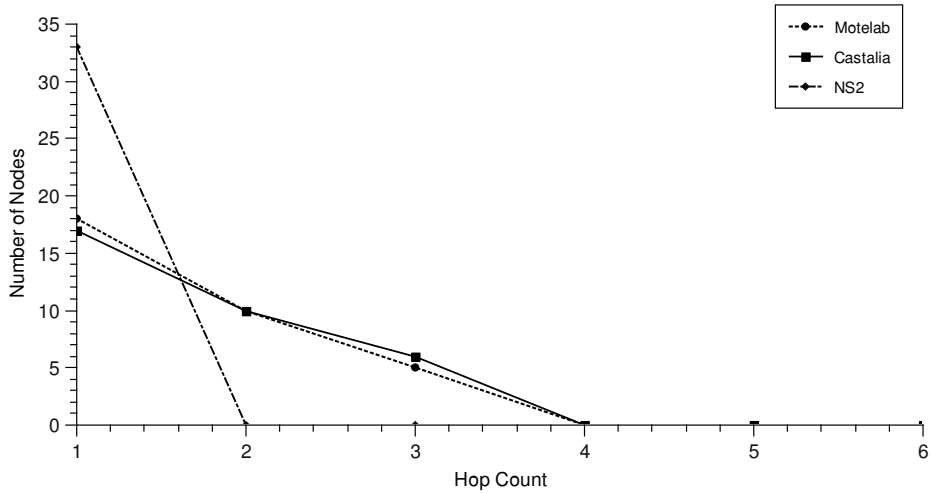


Figure 3.18: Topology Accuracy - *Tuned*, [Ne1]

Surprisingly, in the same configurations, NS-2 always produces a star topology, namely all the nodes are at single-hop distance from the sink. This is a strange behavior, that does not fit the actual topology of the testbed. We made the same experiments in NS-2 varying the value of RS and we obtained results comparable to Castalia's ones for $RS = -85DBm$. We defer a further investigation on this issue to future work, here we limit to observe that this is another argument that discourages the use of simulators as black-boxes.

When [Ne3] is considered, Castalia well reproduces the testbed topology: the maximum hop count is similar, and in the Tuned configuration, Castalia's behavior improves. Also in this case, NS-2 exhibits an unexpected behavior. In the default configuration, all nodes are again at hop count = 1 from the sink (see figure 3.19). When the tuned configuration is adopted, some nodes at distance (i.e. hop count) 2 and 3 starts to appear, but the maximum hop count in motelab is 5 and the overall behavior of NS-2 is still unsatisfactory. These experiments confirm that Castalia can give more accurate results than NS-2, in particular if a tuning of parameters is performed.

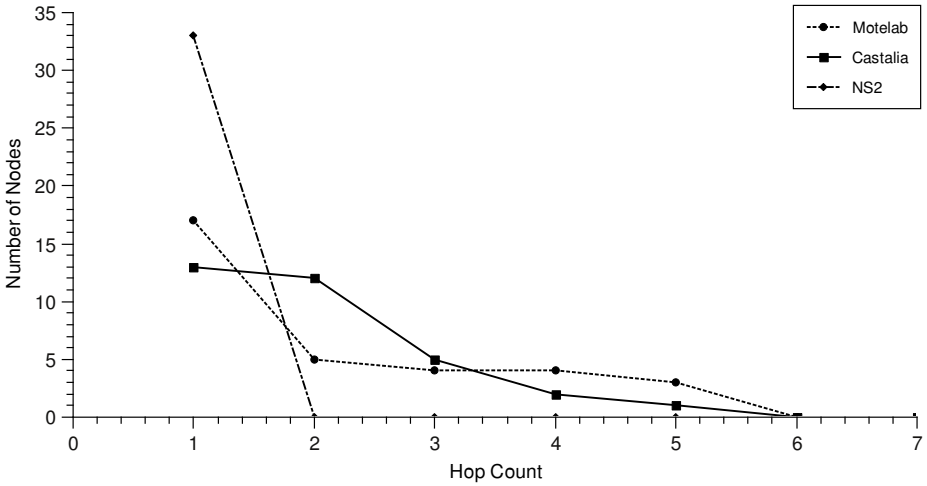


Figure 3.19: Topology Accuracy - *Default*, [Ne3]

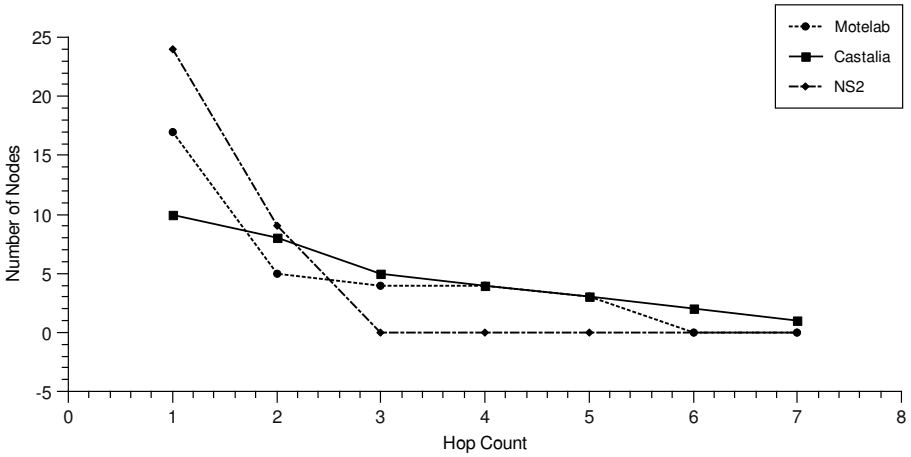


Figure 3.20: Topology Accuracy - *Tuned*, [Ne3]

Packet Delivery Ratio (PDR) In section 3.4.1.1 we have seen that the interference models adopted in the two simulators are quite different. Castalia provides more advanced and refined models. This remark, together with the results presented in the previous sections, might suggest that Castalia provides more accurate results for packet delivery ratio too, but surprisingly, we will see that this is not the case. When the data-rate is relatively low (see figure 3.21), NS-2 and Castalia with the *Additive Interference Model*, produces very accurate results. As expected, the PDR accuracy increases as the link threshold T increases and consequently only more stable and reliable links are considered for message forwarding (see protocol 3.16). We observe that the good performance of NS-2 are quite surprisingly in spite of the fact that its topology accuracy was not very accurate. Moreover, the results in figure 3.22 are even more remarkable; in this case the data-rate is 50kbps and NS-2 outperforms Castalia. Furthermore, considering Castalia, when the ideal modulation is assumed (BER=0), as expected the more refined additive interference model produces the more accurate results. When a PSK modulation is considered (the modulation actually used in Tmote SKY nodes), for high values of the threshold, the additive interference and the simple collision models produce similar results in magnitude but as expected the former tends to over-estimate and the latter under-estimate testbed results. This behavior can be partially explained by the fact that the additive interference model , even if much more accurate than those offered by NS-2 can be still improved. For example, the interference during a packet reception is not constant and currently the maximum interference as the one applied during the whole packet transmission is taken. Also, when multiple transmitters are considered, their powers are simply summed up as we would do if we had white noise. In reality, their aggregate power is usually somewhat smaller than the sum of powers and this depends on the modulation scheme. A further improvement of this model, taking into consideration the above observations, is expected with the next release of Castalia.

3.4.1.4 Final Considerations

We started this section motivating the reasons why spending some time on the analysis of different simulators is crucial to select the one that best fits our requirements. We then presented a detailed analysis of 2 simulators, with particular attention to the WSN simulation; the reason why we conducted such an accurate analysis for these specific simulators is that in an initial phase of our investigation on social networks, using and testing WSN sensor nodes for detecting social relationships was the best solution. Thanks to the technological news, new and best fitted platforms were produced, like the OpenBeacon smart-tags we described in the previous chapter, and this influenced our decision about what simulator was the best one to be used; in this case NetLogo resulted as the best choice. As for Shawn, we explained that our original idea was to use the same source code on iSense devices and on Shawn simulator; unfortunately, iSense devices were not completely appropriate to our experiments, so we used TmoteSky

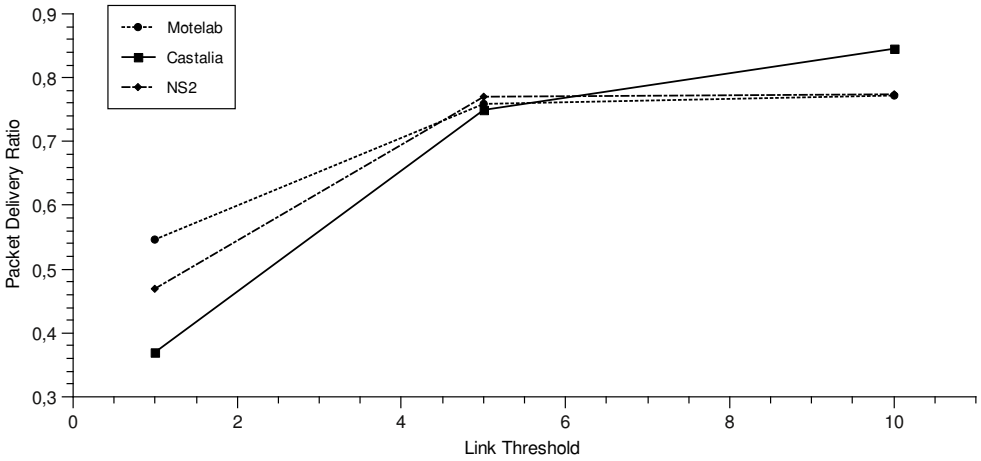
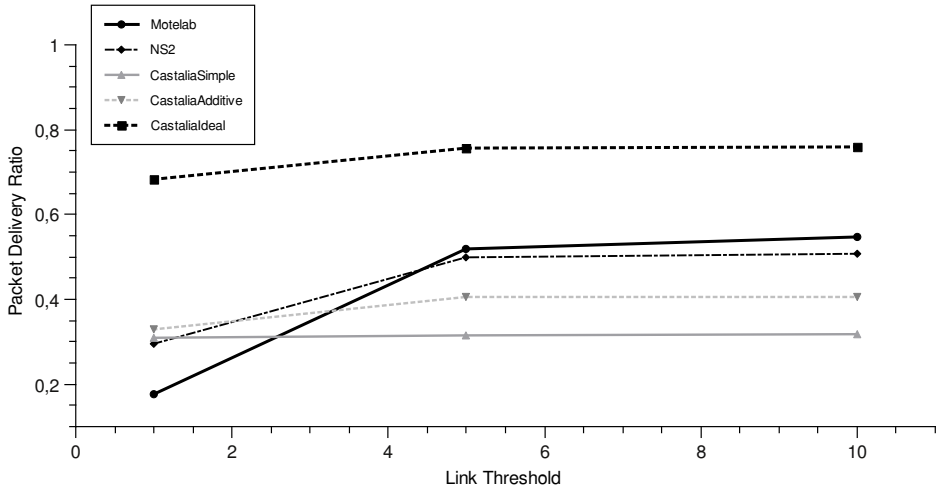
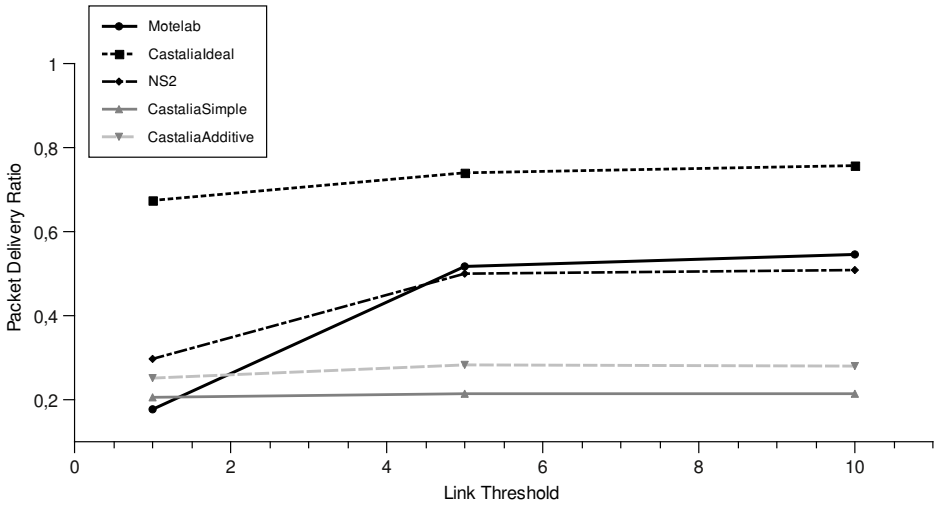


Figure 3.21: Packet delivery ratio - 5kbps

sensor nodes but we decided to keep the same results on simulations obtained from Shawn.



(a) Ideal modulation



(b) PSK modulation

Figure 3.22: Packet delivery ratio - 50kbps

4

Computation and Applications On Dynamic Social Networks

Introducing this thesis, we remarked the growing importance of social and dynamic networks. We devoted chapter 2 and 3 to an overview of theoretical models for representing dynamic networks, to a discussion on the generation of movement traces and to a survey of the currently adopted technologies for recording real traces. These ingredients are important for us for an additional step, i.e. understanding what computations could be executed on this kind of networks. A special attention will be devoted to population protocols, the computational model we decided to investigate in details and that we used in the experiments described in this chapter. Due to their peculiar characteristics, dynamic mobile social networks are extremely interesting for the evaluation of several theoretical problems. In general, dynamic mobile social networks represent a fully-distributed environment, where communication is assumed to be "Opportunistic", and where "Efficiency" plays a key role (as we pointed out in the introduction). Permanent, stable links between nodes in the network cannot be assumed, and nodes are usually devices with very limited resources (in terms of memory and computational capability). Some well-known theoretical problems on distributed systems have been applied to dynamic networks (considering the resulting dynamic graph), each one of them assuming growing features for the nodes or the network itself. We want to demonstrate that even on dynamic mobile social networks it is possible to solve some of these problems. A very simple computational model, with strong restrictions is the *Population Protocol* model. Since we selected this model for a deep analysis for reasons we will explain later, a whole section will be entirely devoted to its detailed description. For now it is sufficient to say that in this specific model, a network is assumed to be composed of devices with $O(1)$ memory (just the space to store a counter), with no identifiers and where interactions are regulated by an external adversary. By assuming simple additional properties on the overall behavior of the network and on devices, some classical problems of distributed computation can be solved. In particular, it was

demonstrated in (84) that the class of predicates that can be computed by this model is the class of semilinear predicates (please refer to the cited papers for the detailed mathematical analysis). It was shown in the same paper that this class can be completely defined by calculating 3 predicates: modulo, threshold and comparison, as we will explain in the following section 4.1. Other interesting problems of computation in fully distribueted systems could be solved in the scenario we envsion. To have an example of the problems we intend, we will refer to the model of Kuhn et al. presented in (16). We introduced the T-Interval connectivity property in chapter 2, but we report it here for the reader's convenience:

Definition 4 *We say a dynamic graph $G = (V, E)$ is T-Interval Connected for $T \geq 1$ if for all $r \in \aleph$, the static graph $G_{r,T} = (V, \bigcap_{i=r}^{r+T-1} E(r))$ is connected. The graph is said to be ∞-interval connected if there is a connected static graph $G' = (V, E')$ such that for all $r \in \aleph$, $E' \subseteq E(r)$*

In their paper the authors demonstrate that the following problems can be solved on a dynamic network were the above-defined property holds:

- **Counting**. Correctly counting the number of nodes in the network.
- **k-verification**. Similar to counting, but each node begins with an input k and when the computation ends, it must correctly output *Yes* or *No* if there are (or not) at least k nodes in the network.
- **k-token dissemination**. An instance of this problem is a pair (V, I) where $I : V \rightarrow P(T)$ assigns a set of tokens from some domain T to each node, and $|\cup_{u \in V} I(v)| = k$. An algorithm solves this problem if for all instances (V, I), when the algorithm is executed in any dynamic graph $G = (V, E)$, all nodes eventually terminate and output $\cup_{u \in V} I(u)$.
- **All-to-all token dissemination**. The same problem as the previous, but here $k = n$ and fro all $u \in V$ we have $|I(u)| = 1$.
- **k-committee election**. Nodes must partition themselves into *committees* (sets), so that the size of each committee is at most k and if $k \geq n$ there is just one committee containing all nodes.

Please note that our focus is not on the resolution of all the problems above, but just to give the reader an idea of how many problems can be considered in a dynamic network. We will omit the detailed descriptions of the algorithms used to solve the different problems and the analysis of complexity, since this goes beyond the scope of our thesis. The interested reader can refer to the original work for additional details. Our focus will be on the *Counting* primitive. Its use is not limited to the context of dynamic networks, but it can be applied to several different contexts of computer science research. In a dedicated section we will describe in details this particular problem. From the examples reported above, it is quite evident how the scientific community is focusing its attention on distributed computation in dynamic networks. For this thesis

we decided to follow a bottom-up approach to present our results. Starting from the most basic technology (in terms of available resources), we show that even with a very low computational power it is possible to implement and solve fundamental problems. The first result we discuss is, in fact, an evaluation and a successive implementation on physical devices of population protocols, a very basic though fundamental theoretical compuational model, on low power devices, i.e. the openbeacon tags we presented in chapter 3. Slightly increasing the power of the devices, we have the possibility to increase the complexity of the problem to solve too. To this purpose, we implemented and tested the *Counting* primitive on telosb sensor nodes, evaluating its performance in terms of accuracy of the estimation. The final step was to develop a more complex algorithm for link prediction and friendship recommendation in a social network of SMS users; we used mobile phones for this application, following the approach of using more complex technologies for more complex problems.
In the final part of this chapter, we will present and discuss an unexpected application of our work on the collection of traces: thanks to a collaboration with a contemporary artist, we realized an artwork that was exposed at MACRO museum of Rome.

4.1 Computation on Networks of Tiny Devices: Population Protocols

According to (85): *"Population Protocols are used as a theoretical model for a collection (or population) of tiny mobile agents that interact with one another to carry out a computation."*. Because of their simplicity and their ability to calculate very basic predicates (as we will describe in the following), we decided to analyze them in more details and to evaluate its behavior in a realistic scenario. Population protocols (86, 87)(we often use the acronym PP in the following) provide a theoretical model of a collection of tiny, possibly mobile agents that interact to carry out a computation. Agents are assumed to be identical finite state machines. Each agent initially possesses an input value that determines its initial state, while pairs of agents can exchange state information whenever they interact. Interaction patterns of the agents are unpredictable, but they are subject to some fairness constraints, so that any computation eventually converges to a stable output. Removing this fairness constraint and studying the behavior of the basic PP when the interaction patterns follow the movement of real persons has been part of this thesis, as we will show in the following. In this section we provide an overview of the basic PP model.

A PP $\mathcal{A}$ is a 6-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{Q}, \mathcal{I}, \mathcal{O}, \delta)$ where $\mathcal{X}, \mathcal{Y}, \mathcal{Q}$ are finite sets and:

1. $\mathcal{X}$ is the input alphabet,
2. $\mathcal{Y}$ is the output alphabet,
3. $\mathcal{Q}$ is the set of states,
4. $\mathcal{I} : \mathcal{X} \to \mathcal{Q}$ is an input function mapping inputs to states,

5. $\mathcal{O} : \mathcal{Q} \to \mathcal{Y}$ is an output function mapping states to outputs, and
6. $\delta : \mathcal{Q}x\mathcal{Q} \to \mathcal{Q}x\mathcal{Q}$ is the transition function on pairs of states.

A computation involves n agents, where $n \geq 2$. Each agent initially has an input value from $\mathcal{X}$. Each agent's initial state is determined by applying $\mathcal{I}$ to its input value. This in turn determines an initial *configuration* for an execution. A configuration of the system can be described by a vector of all agents' states. An execution starting in the initial configuration proceeds in steps, in each of which an interaction between two agents occurs. Assume two agents in states p and q respectively interact. If $\delta(p, q) = (p', q')$, we call $(p, q) \to (p', q')$ a **transition**. Note that, in general, interactions are not symmetric, in the sense that one of the involved nodes acts as the *initiatior*, while the other acts as the *responder*. Also note that the notion of transition immediately extends to configurations in the obvious way. In any configuration, the function $\mathcal{O}$ maps the current state of every agent to an output. A population protocol computes some predicate P defined over the set of possible inputs if, for any given input, the outputs of all agents eventually stabilize to the correct value of the predicate for the input under consideration.

In (86, 88), the authors assume that interactions between agents are determined by an adversary, which is constrained to a fairness condition, ensuring that every computation eventually stabilizes. Stability is defined as a situation where computation reaches a configuration $\mathcal{C}$ after which, no matter how the computation proceeds, no agent will be able to change its output value. $\mathcal{C}$ is then called an **output-stable** configuration. In (86, 88), the authors also propose an alternative mechanism to determine the sequence of interactions, according to which, at any time step, two agents are selected uniformly at random for interaction.

For the basic PP model there exists an exact characterization of the set of computable predicates: these are exactly the semlinear predicates, or equivalently the predicates definable by first-order logical formulas in Presburger arithmetic (84, 86, 89, 90). It has also been shown in (86) that the following three predicates are sufficient to define all those that can be computed by Population Protocols:

- *Threshold Predicate.* For any given $a \in \mathcal{X}$, this predicate is true if $\#a \geq T$ where T is a given threshold[1].
- *Modulo Predicate.* This predicate is true whenever $\#a \equiv j(mod k)$ for given j and k.
- *Comparison Predicate.* This predicate is true whenever $\#a \geq \#b$, for $a, b \in \mathcal{X}$.

We next outline simple algorithms that allow computation of these predicates in the population protocol paradigm. In the following paragraphs, we assume without loss of generality that $\mathcal{Q}$ is an integer interval.

[1] In the remainder, for any $a \in \mathcal{X}$, $\#a$ denotes the number of agents whose input value is the symbol a.

Threshold predicate. In this case, $\mathcal{Q} = \{0, 1 \ldots, T\}$. The transition function is the following:

$$\delta(q_1, q_2) = \begin{cases} (q_1, T), & q_1 = T \\ (q_1, q_2 + 1), & 1 \leq q_1 < T \, and \, q_1 = q_2 \\ (q_1, q_2) & otherwise \end{cases}$$

The input function is $I(x) = 1$ if $x = a$, $I(x) = 0$ otherwise. Finally, the output function is $O(q) = 1$ if $q = T$, $O(q) = 0$ otherwise.

At the onset of the execution, agents with input a are in state 1, while all other agents are in state 0. During execution of this protocol, all but one of the agents in state i advance to state $i + 1$ in each case. This "tower" will extend to T if and only if there are initially at least T agents in non-zero states.

Modulo predicate. The algorithm to calculate the modulo predicate is quite similar to the previous one. The initiator becomes passive while the responder remains active and keeps the combined data value. Eventually, exactly one agent will be active and will store the correct sum modulo k, and it will communicate the correct output to all of the other agents. As a concrete example, we specify a protocol over the alphabet $\{a, b, c\}$ that computes whether the number of occurrences of a in the input is congruent to 2 modulo 3. The states of the agents are of the form (x, y), where $x \in \{A, P\}$ indicates whether the agent is active or passive, and $y \in \{0, 1, 2\}$ is a data value modulo 3. The input map I maps the input symbol a to the state $(A, 1)$ and the input symbols b, c to the state $(A, 0)$. The output map O maps the state (x, y) to 1 if $y = 2$ and to 0 otherwise. The transition function is defined as follows, where $+$ denotes sum modulo 3. Note that the initiator's state is updated independently of the responder's state. Transition function:

$$\delta\{(x1, y1), (x2, y2)\} =$$

$$\begin{cases} ((P, y1), (A, y1 + y2)) & x1 = A, x2 = A \\ ((P, y1), (A, y1)) & x1 = A, x2 = P \\ ((P, y1), (x2, y2)) & otherwise \end{cases}$$

Comparison predicate. Given a population of agents tagged with a or b, we want to know if $\#a \geq \#b$. The output function will be $Y = \{0, 1\}$, with 1 indicating that there are more a's than b's. The procedure adopted by this algorithm is a cancellation procedure: whenever an agent having an a encounters an agent having a b, they both cancel themselves. At a certain point, only agents with a's (if $\#a > \#b$) or b's (if $\#a < \#b$) will remain. The input function is the identity, the set of states is $Q = \{a, b, 0, 1\}$ and the alphabet is $\sum = \{a, b\}$. The transition function is thus defined as follows:

$$\delta = \begin{cases} (a,b) \to (0,0) \\ (a,0) \to (a,1) \\ (b,1) \to (b,0) \\ (0,1) \to (0,0) \end{cases}$$

The first rule ensures that, eventually, either no a's or no b's will remain. At that point, if there are a's remaining, the second rule ensures that all agents will eventually produce output 1. Similarly, the third rule takes care of the case where b's remain. In the case of a tie, the third rule ensures that the output stabilizes to 0.

4.1.1 Population Protocols Variants

Although we didn't use the following models in our experiments, we decided to include a short description of some variants of the basic population protocols model analyzed by the authors in (91) for the sake of completeness. We omit formal description, that the interested reader can find in (91).

Transmission with queuing This model differs from the basic one in the way the agents interact: instead of having a direct interaction with each other, they communicate by sending messages. Moreover, an agent is not aware of the current state of the other one and the identity of an agent cannot be deduced from a message. To grant the full power of this model (meaning the same computational power of the basic PP), messages are not sent by the sender agent to a specific receiver, but they are "launched" and put in a special multiset of messages in transit, until a receiver agent receives it. We remark that we are giving just a very short description of this and the following models, this explains why formal definitions and demonstrations are omitted.

Immediate Transmission In this case we are considering a weaker model, where the communication is one-way (i.e. the initiator does not learn the state of the responder). For this reason, the state of the initiator is updated independent of the state of the responder and thus the transition to a new state can take into account just the fact that an interaction (or a message sending) has taken place, but no information is available about the current (or the new, after the interaction) state of the responder.

Immediate Observation This is a small variation of the immediate transmission model: in this second case the initiator is unaware of the fact that it is "observed" by the responder, and thus it has not an occasion to update its state.

Delayed Transmission This version is a special case of the queued transmission model, but in this case the recipient agent does not have the occasion to refuse of receiving a message. For this reason, the recipient coul be overwhelmed by incoming messages, and this fact limits the power of the model.

Delayed Observation In this case, a special variation of delayed transmission, an agent cannot refuse an incoming message and cannot update its state after sending a message. This is the weakest population protocol model analyzed by the authors.

4.1.2 From theory to practice: Population Protocols on Real Social Networks

In order to evaluate the real possibilities of a very basic theoretical model like the Popoulation Protocols we described so far, we decided to perform 2 types of experiments. In the first one, we used the traces we collected in our University and at the Macro museum (see chapter 3 for details) to feed the simulator. The idea was to make a first step in the direction of making the context of the experiment more real by using real movement traces rather than a completely random model. In the second experiment we implemented the whole population protocol model on the very simple devices we described in chapter 3, the OpenBeacon USB 2 tags, to verify if it is possible to use such a theoretical model in a real physical context and what is the overall result.

4.1.2.1 An initial step: Population Protocol Evaluation using Real Traces

As discussed in chapter 3, we used OpenBeacon Tags to collect and record real movement traces from a population of volunteer participants. In this section we describe how we used these traces to test the behavior of population protocols when interactions are not randomly selected but based on the information we recorded in reality (92), (93). In our experiments we implemented the population protocols in Netlogo (73), a multi-agent programmable modelling environment extremely useful to analyse at high level the behaviour of a population of agents. Each agent in a Netlogo simulation runs a PP program (i.e. a Finite State Machine) and is uniquely associated with a real user. This means that the agent moves according to the trace of the real user collected in our test-beds. In our simulation, we are interested in studying the convergence time of PPs when interactions between agents are determined by the real movement of the agents as recorded in our data-logs. The *convergence time* is the time required for all agents to stabilize on the same correct output.

Simulation setup The durations of the two testbeds are very different: the DIAG testbed last 5 days, corresponding to 283611 timestamps, while the MACRO testbed last ≈ 3 hours and contains 8634 timestamps. To strike a balance between efficiency of the simulation and dynamicity of the interaction graphs, and in order to normalise results of the two different data-logs, we limited the number of interaction graphs considered in the experiments to 4317 (i.e. 8634/2). As described before, the data-sets contain an interaction graph for each timestamp t_i. In the MACRO case, we considered an interaction graph $G_i = G^M_{2i-1} \cup G^M_{2i}$ for every $i = 1, 2, .., 4317$, where G^M_i is the original interaction graph at time stamp t_i in the MACRO (M) testbed. Similarly, for

the DIAG testbed we considered the interaction graph $G_i = G^D_{66i-65} \cup ... \cup G^D_{66i-1} \cup G^D_{66i}$, where $i = 1, 2, .., 4317$ and G^D_i is the original interaction graph at time stamp t_i in the DIAG (D) testbed. Note that each interaction graph corresponds to 2 consecutive timestamps (and original interaction graphs) in the MACRO experiment and 66 in the DIAG one. In our simulations, given a population of 120 nodes, we first analyse the performance of PP when the probabilistic adversary is used; in this case the couple of agents that interact is chosen uniformly at random among the whole population. Then, we analyse the performance of PPs when the interactions are driven according to the interaction graphs G_i. In particular, at each step of the simulation, there is an interaction between a randomly selected node in G_i and one of its neighbours. Convergence time is measured in terms of simulation steps necessary to stabilize.

An important point to notice is that the number of steps necessary to converge may exceed the number of available timestamps in the data-sets. For this reason, whenever this condition occurs, a new round of the simulation begins, in which the whole sequence of interaction graphs is again presented to the algorithm. Note that the execution in the new round is not going to be the same as in the previous one, because interaction graphs only determine the set of possible interactions, whereas the actual ones are the result of a random process.

We repeated 5 executions of each set-up for each different predicate, and we obtained the results described in the next paragraphs by averaging over these executions.

Modulo and Threshold Predicates Results for these experiments are summarized in Figures 4.1, 4.2 and 4.3. In these figures, the x-axis corresponds to simulation steps. For a given number x of simulation steps, the y coordinate gives the number of nodes (among the 120 considered) that have converged to the correct output within x steps. As can be noticed by comparing the 3 graphs, convergence is faster on the random topology for both predicates, with modulo requiring a bit more steps to converge. This is an expected behavior and is due to the fact that in this case the underlying topology is a complete graph, thus at each step, every node can be selected to interact with every other node in the network. The same consideration does not apply to the real traces, where a node can be selected to start an interaction only when it is actually active in the current timestamp and furthermore it can interact only with one node in its current neighbourhood.

Convergence is reached in both the DIAG and MACRO cases. In particular, for the DIAG testbed convergence is achieved after $\approx$ 800 thousand and 1.8 million steps for threshold and modulo respectively. In the MACRO testbed, these numbers decrease to $\approx$ 100 thousand and $\approx$ 140 thousand respectively. However, it is worth nothing that in both cases, a significant fraction of nodes converge to the correct output well before the final convergence time. This means that, while in the considered cases all nodes eventually converge to a stable output, the time required by the whole population to stabilize (i.e. the convergence time) strongly depends on the interaction patterns. This is due to the fact that the few nodes that still have to converge, are selected for

interaction with low probability because they are likely the nodes with lowest (average) degrees. In other words, those are the nodes that have less interactions with other nodes and thus less opportunities to converge to the final state. We recall that we removed isolated nodes, namely nodes that did not have any interaction with other nodes, as they obviously never take part in the computation.

We finally remark that although normalized, the MACRO testbed converges about an order of magnitude faster than the DIAG one. This behaviour can be explained by the different density in space and time of the two testbeds. In particular, in the MACRO testbed most of the visitors were in the same room for the short amount of time of the visit, while in the DIAG testbed, the space available for users was bigger and the timespan of the experiment was 5 consecutive days. This is confirmed by the fact that in the DIAG experiment, the total number of contacts is 63296 over 283611, whereas in the MACRO experiment, it is 48263 over 8634 timestamps. Therefore the "contact density" is significantly higher in the MACRO experiment.

Comparison Predicate The comparison predicate is "harder" than modulo and threshold predicates, essentially because (as was shown in (88)) this predicate can be calculated only by two way population protocols: this means that the state transition of the initiator agent depends on the current state of the responder.

Recall that in this case, given a population of agents tagged with a or b, we want to know if $\#a \geq \#b$. The output function will be $Y = \{0, 1\}$, with 1 indicating that there are more a's than b's. The procedure adopted by this algorithm is a cancellation procedure: whenever an agent having an a encounters an agent having a b, they both cancel themselves by setting their value states to 0. A value state of 0 is equivalent to a cancellation because an agent "a" can get this value if and only if it interacts with an agent "b". At some point, only agents with a's (if $\#a > \#b$) or b's (if $\#a < \#b$) will remain.

The original definition of this predicate states that the computation converges when all the agents output the correct answer, i.e. all of them are in the states $\{0$ or $b\}$ or $\{1$ or $a\}$. We slightly relaxed this condition, essentially because, due to the particular characteristics of our social topologies (quite sparse graphs) convergence was not obtained in a reasonable time according to this definition, even though the protocol essentially completes its computation much earlier. In practice, we used an "external observer" that stops the simulation whenever either no a or no b remain in the population. When this condition occurs the protocol has essentially completed its computation. This follows since the comparison protocol adopts a cancellation procedure, so that when no agents of one kind or the other remain, the aggregate information of the population already allows to answer the predicate and eventually every agent will stabilize to the correct output.

We also introduced a *State Swapping* procedure, following the idea introduced in (86) to address the problem of restricted interaction graphs. The State Swapping procedure simply consists in exchanging the states between two agents when they interact.

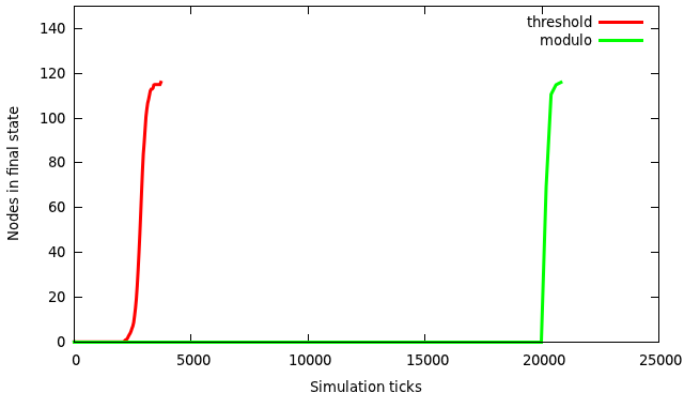


Figure 4.1: Modulo and threshold: random topology

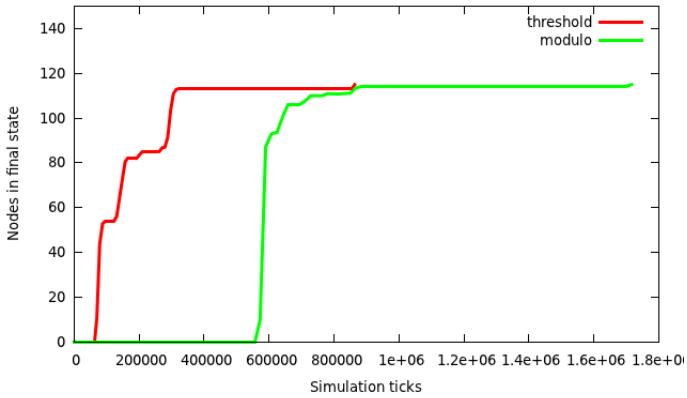


Figure 4.2: Modulo and threshold: DIAG social topology

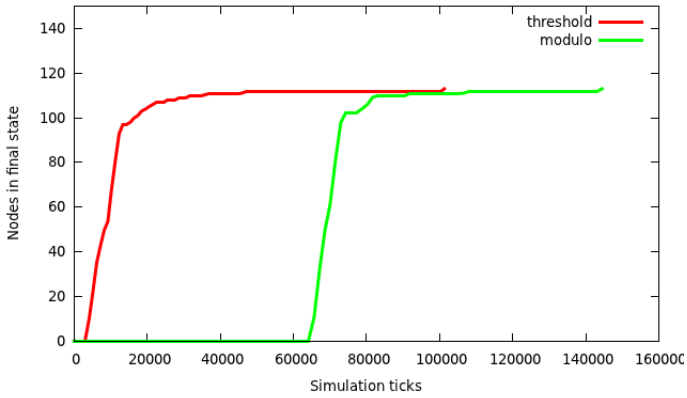


Figure 4.3: Modulo and threshold: MACRO social topology

The interaction graph does not make the model any weaker, since adjacent agents can swap states to simulate free movement. In our case, this procedure allows nodes with lower degree (i.e. nodes with weak interactions) to simulate a free movement and thus to be more likely involved in the next steps of the evaluation of a predicate. In our experiments we used state swapping probabilities 0.1, 0.25, 0.5 and 1 in order to test how different probabilities affect convergence. We stress that without state swapping, the comparison predicate often does not converge. This happens because this predicate is more sensitive than others to sparse networks, particularly when the two quantities to compare are very close. In this case, the cancellation procedure occurs with lower probability as computation proceeds. When $\#a$ and $\#b$ are close, cancelling the few remaining a's (or b's) requires the corresponding nodes to be selected uniformly at random as part of the current interaction graph G_i, which can be a low probability event. Conversely, when $\#a \gg \#b$ or viceversa the comparison predicate converges also without state swapping.

In the following Figures 4.4, 4.5, 4.6, the y-axis represents the number of remaining pairs (a, b) still in the population (the x-axis is again the number of simulation steps). This is an indirect indication of the number of interactions that must still occur until convergence (according to our "relaxed" termination condition). As can be noticed, the random topology still provides faster convergence, but increasing the swapping probability, the convergence time of the PP on the real topologies can be improved. Indeed, as in the earlier predicates of modulo and threshold, we can observe a faster convergence of the comparison in the MACRO experiment (fig. 4.6) with respect to the DIAG experiment (fig. 4.5). This behaviour is clearly expected for the same reasons of the other two predicates. Nevertheless the significant aspect to observe is the dizzying convergence curve's descent for around 80% of the total agents in both of cases. This percentage seems to be slightly higher in the random case (fig. 4.4).

In the work we presented so far, we used the interaction data we recorded to create real contact graphs on an agent simulator (NetLogo) to compare the behavior of population protocols on a random topology (as done in the original work) to those obtained when using our realistic contact graphs. Results showed that even though the full convergence time is greater in case of the realistic mobility (depending on the density of contacts, it can be lower or faster, but the random topology is always the fastest performer), the vast majority of agents ($\approx 80\%$) converge well before and in any case the population protocols work fine even with a mobility model taken from reality. Next step is thus to implement the population protocol model on physical devices and evaluate their behavior.

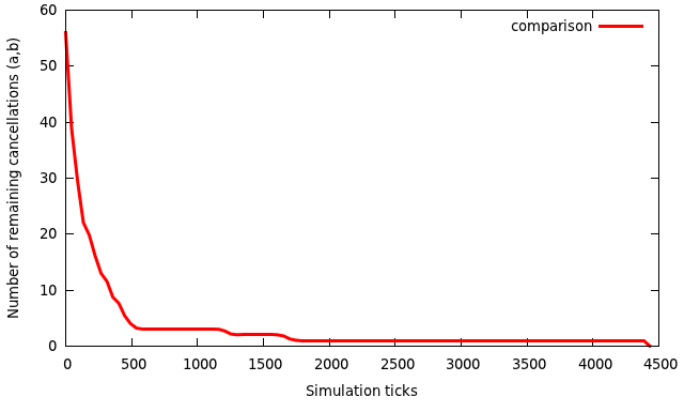


Figure 4.4: Comparison: random topology

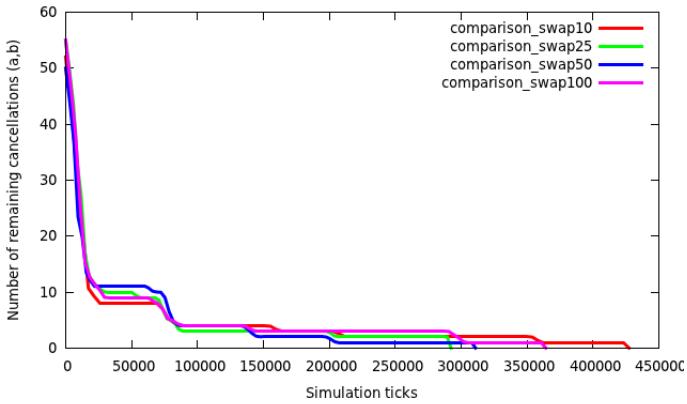


Figure 4.5: Comparison: DIAG social topology with different swapping probabilities

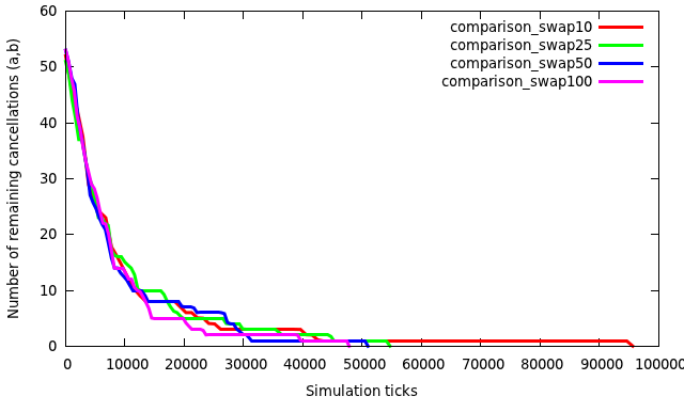


Figure 4.6: Comparison: MACRO social topology with different swapping probabilities

4.1.2.2 Population Protocol Implementation on Physical Devices

The wireless devices we used for the real implementation are the OpenBeacon tags 3.5 we described in the previous section; they reproduce the constraints of the theoretical PP model in many respects, namely they have very limited computational, storage and communication capabilities and they allow a fine-grained detection of social interactions. Our main goal was investigating the behaviour of a prominent theoretical model when implemented and executed in a real setting, and to evaluate technical problems, assess performance and practical limitations of the protocol (94). The generic architecture for data collection is the same presented in section 3.2, so we will focus our attention on the implementation details. As explained in previously, in the PP model, at each step, the adversary selects an initiator and a responder that interact with each other and then change their local states according to the PP protocol. In a real distributed setting, there is no coordination among nodes, namely we cannot assume there is a single (initiator, responder) pair at each step. This is not a problem if the couples of interacting nodes are fully disjoint, but if this is not the case, the system could end up in an inconsistent state as shown in the scenario depicted in figure 4.7.

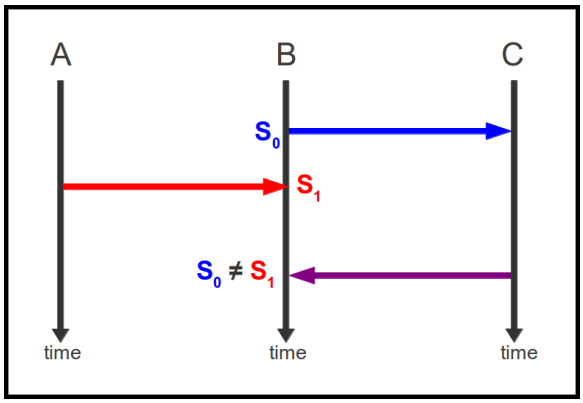


Figure 4.7: A critical scenario

When node B initiates an interaction with C it is in state S_0 . Before completing its transaction, it becomes the responder of an interaction initiated by A and consequently updates its state to S_1 . As node B eventually receives the message from node C that allows it to complete the transaction it initiated, its state has changed and this can lead to inconsistent configurations of the system. In order to address such scenarios, it is necessary to implement a mechanism that forces a node to complete a transaction before being involved in the next one. Below we describe the protocol we implemented to this purpose. The protocol consists of three phases:

- **Phase 1 - Setup** When nodes boot, they choose a random number *Rnd* in the range $(0..X)$. Randomness is granted by combining the local time of a node with

the node ID[1] as a seed for the random number generator. Note that the local timer is started by each node once it is re-programmed through the USB connection, and it is never stopped. By programming nodes at different times and by combining their local times with their unique identifiers to obtain initial seed values, we can ensure that most likely the random numbers selected by the nodes are different from each other. This is important for the initiator election phase described in the following. In particular, the node chosen as the initiator will be the one with the highest value of Rnd, and it will be the initiator of the 3rd phase of the algorithm.

- **Phase 2 - Neighbour Discovery and Initiator Election**

  In the second phase each node starts to periodically broadcast messages containing its ID and the Rnd value generated in phase 1. Every node maintains a local list of neighbours and an "initiator" flag that is initially set to true. Every time the node receives a message (ID_x, Rnd_x) from a neighbour x, it performs the following actions:

  – if not present in its local list of neighbours, the receiver node adds neighbour x to this list;

  – the receiver node compares Rnd_x to its local value Rnd: if $Rnd_x > Rnd$, the receiver node sets its local initiator flag to false, meaning that for the current iteration of the algorithm it cannot be an initiator in phase 3; if $Rnd_x = Rnd$, then an additional check is performed on the received ID, choosing as possible initiator the node with the highest ID; if $Rnd_x < Rnd$, then the receiver node can still be chosen as an initiator, and its initiator flag remains set to true.

  For the sake of simplicity, in our experiments we considered a fully connected network. As a consequence, at the end of phase 2 of every iteration, each node should know its neighbours, and only one node should be the initiator for the phase 3.

- **Phase 3 - Initiator-responder interaction**

  In this phase, the initiator randomly selects a responder among its neighbours and interacts with it according to the PP protocol.

For testing our implementation we asked 10 volunteers to carry a device while randomly moving inside a room; in the meanwhile a Reader to collect traces and debug information was placed in the same room and connected to a notebook to analyse the data. In this scenario, the visibility graph (i.e. the graph in which there is a link if two tags can communicate) is fully connected. We stress that also in this simplistic network scenario, several interesting issues arise.

The trace files regard two different types of messages: **Contact messages**, containing the neighbors each node met during the ND phase, and **Population messages**,

[1]Note that, though ids are not assumed in the standard population protocol model, they are consistent with the capabilities of current devices and their presence has a negligible impact on storage requirements.

giving us information on the behaviour of the population protocols(e.g. what tag is elected as initiator, if the computation is still going on and similar debug information) The traces are also used to run the PPs on the simulator NetLogo in order to compare the results obtained on real devices to those generated by the simulator. Even though population protocols are not guaranteed to converge, but only to stabilize to the correct output, the time taken in practice to the protocol to reach its final configuration is clearly important in a real setting. For this reason we study the behaviour of our implementations of population protocols with respect to it.

In our experimental settings, in which for the sake of simplicity we considered a fully connected visibility graph, the neighbour list of each node should contain all the other ones. However, we soon realised that this is not the case in our testbed. Indeed, because of possible interferences or collisions during transmissions, nodes end the Neighbour Discovery phase with a partial knowledge of their neighbours.

To better investigate this phenomenon, we varied the duration time of the Neighbour Discovery in the range $\{200, 1000, 5000\}$ milliseconds. Because nodes periodically broadcast their id, the longer is the ND phase, the higher is the likelihood of discovering all the neighbours. The results of this experiments are shown in figure 4.8 in which bigger and darker nodes indicate a higher number of distinct neighbours, while the thickness of edges is proportional to the number of messages received from a neighbour. As expected, increasing the duration of the ND phase results in a more connected topology, even if also for a duration of 5 seconds, two nodes are not connected with all the others.

To better clarify the effects of a partial knowledge of the neighbors, let us consider the scenario depicted in figure 4.9. In this case, nodes A and C are not neighbours. At the end of the initiator election phase, according to the protocol described before, both A and C are initiators (they have a Rnd greater than B one) and can thus start an interaction with B. In case of a simultaneous interaction we have an instance of the classic hidden terminal problem. Unfortunately, the current implementation of the MAC cannot deal with such problem.

In our experiments, due to the presence of the hidden node terminal and more in general due to interferences and collisions, we cannot assume the absence of link-failures, and thus the convergence of standard population protocols cannot be assured. In (95) the authors presented a technique for dealing with link-failures. In this work we are interested in evaluating the effects of practical problems on the standard model and we plan to investigate the performance of fault tolerant versions of population protocols in future work.

However, depending on the duration of the ND phase, we can expect at each step a different network topology with different levels of connectivity. In this situation, the threshold predicate can converge in any case. Indeed, partial results computed in possibly disconnected networks, can contribute to the final result when some specific nodes can eventually communicate. To better clarify this concept, we consider the example in figure 4.10. Assume we want to compute the threshold predicate with

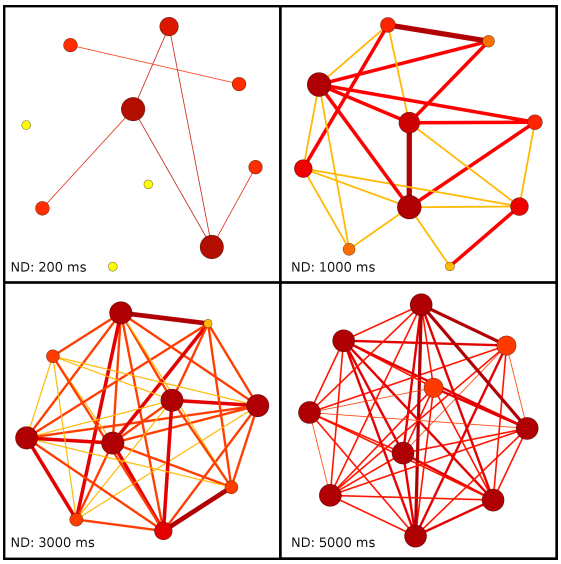


Figure 4.8: Network topology varying ND length

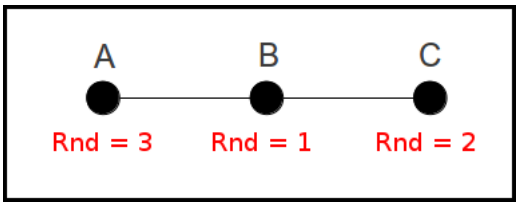


Figure 4.9: Effects of the partial knowledge of neighbours

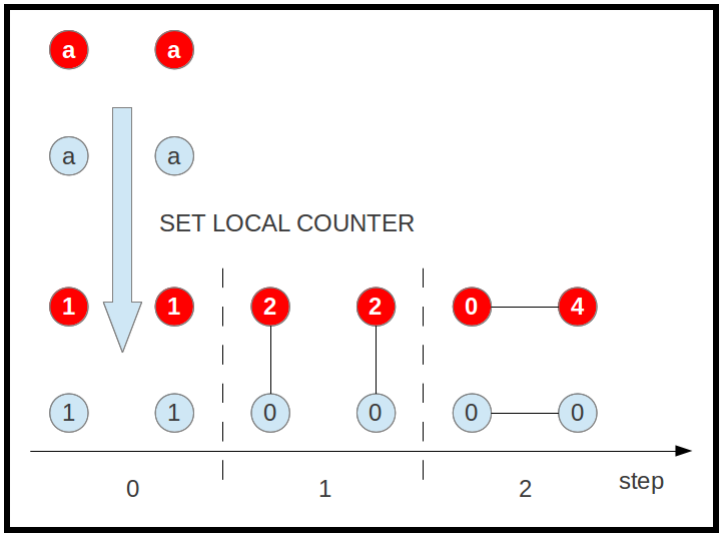


Figure 4.10: The threshold predicate converges even in presence of link-failures. In this example, the protocol verifies if $\#a > 3$

threshold equals to 3. At step 0 all nodes are initialised with 1. At step 1, the network is partitioned in two sub-networks and after the interactions (i.e. phase 3) a node in each of the sub-networks has changed its counter (i.e. status) to 2, while the other has its counter to 0. At step 2, other two sub-networks are formed. One of the node is above the threshold and can then propagate this information: the convergence time cannot be guaranteed, but the predicate will eventually converge.

In the case of comparison and modulo predicates, in presence of link-failures, the convergence cannot be guaranteed. The computation of such predicates requires two phases: in the first phase all nodes participate to the computation, but the final result is at the end stored in a single active node. In the second phase, at each step the active node becomes passive and propagates the result to another node which becomes the new active node. The second phase, terminates when all the nodes have been contacted. Observe that, if during this phase there is a link-failure, the new active node cannot be elected. As a consequence, all nodes in the network are in passive mode and the procedure cannot proceed.

For the above reasons, in the following of the paper we will focus on the threshold predicate comparing the result obtained on real devices with those obtained simulating the protocol on the NetLogo simulator using the traces collected in the real experiments.

The results on real devices are shown in figure 4.11. As expected, the protocol with the longer Neighbour Discovery (ND = 5000) converges before. This is because the graph on top of which the protocol runs is more connected (see figure 4.8) and thus the initiator can possibly interact with any of the other nodes at each step. This is not the case when the connectivity decreases, and thus not all interactions are possible at each step (see figure 4.10). However, if instead of considering the number of steps, we

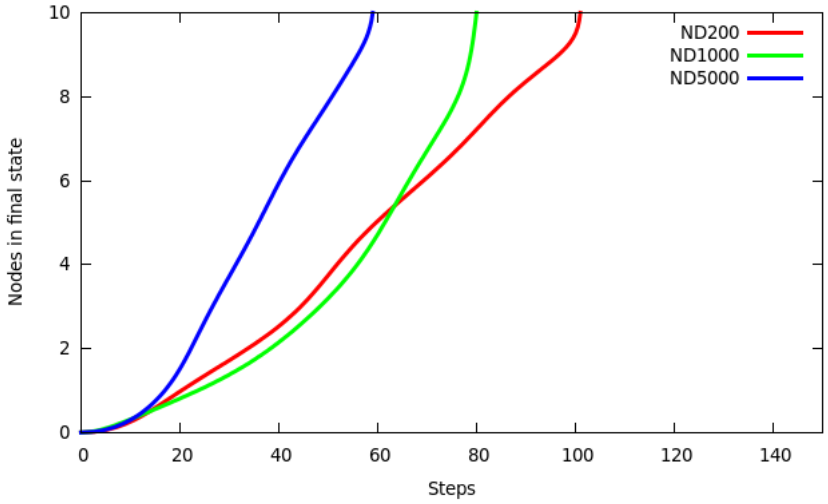


Figure 4.11: Threshold results on real devices varying ND length

consider the time necessary to converge, the results change dramatically. Indeed, the protocol with the smaller ND duration takes 20000ms to converge ($ND200 * 100steps$) while the one with the longer ND duration takes 30000ms.

In figure 4.12 we show the results on the same protocol run on a simulator having as input the traces collected in our real experiments. In this context, we use reliable links, namely there are not link-failures. The results show similar trends as the real ones, namely the protocol with the longer ND converges earlier in terms of steps, even if with a steeper behaviour. However, the steps necessary to converge are remarkably higher than those observed in the real experiments. This can be explained again observing the figure 4.10 and recalling that NetLogo is a discrete event simulator in which at each step only a single interaction is possible. If we consider step 1 of figure 4.10, in the simulation case only a single couple of red-blue nodes can interact, while in the real case they can interact simultaneously. In other words, in calculating the threshold predicate with real devices we take advantage of some kind of parallelism which is not available in the simulator.

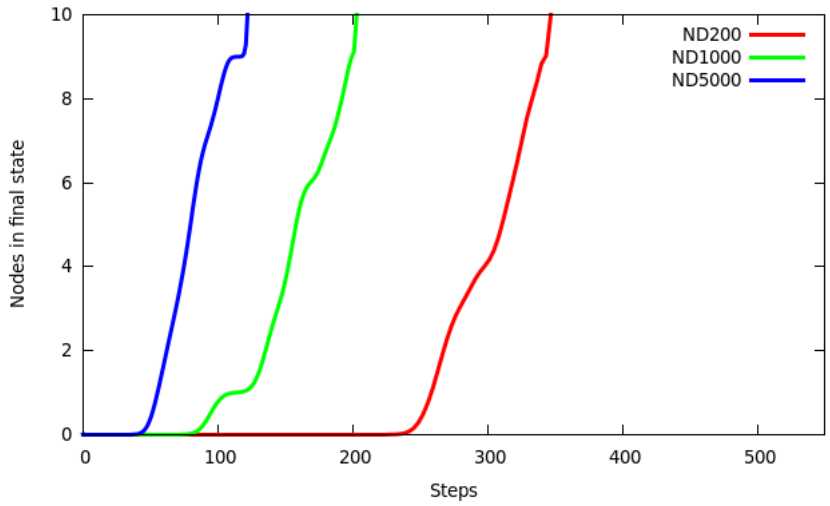


Figure 4.12: Threshold results on simulator varying ND length

4.2 How many are we? The Distinct Counting Problem

Introducing this chapter, we pointed out how estimating the number of distinct elements in a set is a relevant and difficult problem in different areas; when the considered set is composed of mobile entities the problem is even harder. In the view of testing theoretical models and algorithms in real contexts, we decided to focus our attention on the distributed distinct counting for its relevance in several important areas of computer science. After an introduction of the problem itself and of the various areas of its appliability, we will present some of the algorithms proposed in literature for an efficient estimation of distinct elements in distributed settings. We will close this section with a description of our implementation of one of these algorithms and of our experiment of evaluation in a realistic context. The basic computing primitive we want to introduce in this section is *DISTINCT*, i.e. an estimate of the number of distinct elements in a set/stream/population/ecc. Applications of this primitive range from databases to data stream analysis, and below we review some of the main research fields where it is useful. We will also present some of the most diffused techniques to cope with this issue, and we will describe in details the one we chose and applied to monitor a population of individuals.

Databases Historically, this was probably the first field were an efficient distinct couting estimation became a necessity. Counting the number of distinct elements in columns or in a table is a basic problem in databases; this has applications to estimating the selectivity of queries, or in query optimization, as explained in (96) or in (97). As the average dimension of databases was exponentially growing, maintaining each single element in an array or in every other data structure requiring linear memory

was unfeasible.Thus, new techniques to estimate the number of distinct elements in the collection were necessary. Starting from this necessity, Flajolet and Martin in (98) presented a probabilistic technique based on hash functions to efficiently estimate the number of distinct elements in the considered set; we will discuss this work in a dedicated subsection, as it was the basis for many of the successive distinct counting algorithms like (99) and in particular for our implementation of distinct counting on mobile tiny nodes.

Aggregation Techniques in Wireless Sensor Networks Authors in (100) report the deployment of wireless sensor networks in a wide range of applications like environmental or traffic monitoring, smart homes, fire detection and structural integrity monitoring just to name a few. In the above scenarios, individual values are usually not relevant: users are more interested in the quick extraction of useful synopses about a large portion of the underlying observation set; in other words we would like to be able to efficiently answer queries like "What was the average temperature over the entire field during the last 12 hours?" Trying to collect every data monitored by each individual sensor would be unrealistic in terms of resource usage, so the canonical approach is to compute statistical aggregates, such as maximum, minimum, average, etc., to summarise the distribution of data. Since this information must be extracted and combined across multiple locations and devices, repeatedly and in a dynamic way, in-network aggregation schemes (101) must be developed to efficiently merge and quickly update partial information to include new observations. Obviously, to efficiently aggregate data amd to compute statistics like for example the average value, it is necessary to keep track of the number of distinct elements that contributed to the overall statistics. If we consider the left figure 4.13

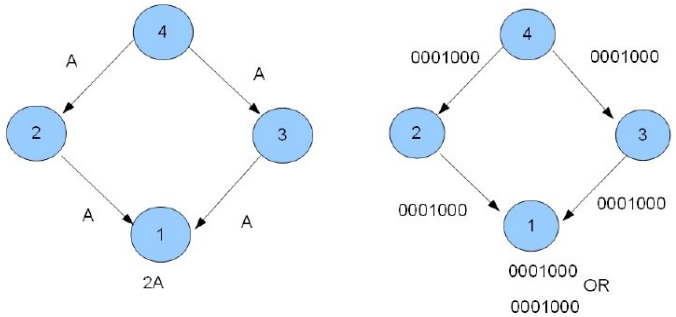


Figure 4.13: The problem of duplicate data

we have a clear example of why an accurate estimation of distinct elements is needed: the data generated from node number 4 (indicated as 'A') is propagated to both nodes 2 and 3; they both transmit this reading to the root node 1, that will receive both data and, being unaware of the duplicate value, it will consider 2 different readings, thus it

will wrongly count '2A'. A first trivial solution consists in sending from each node a bitmap in which each position contains a value, so that when node 1 receives the same data $[i]$ from nodes 2 and 3 only position $[i]$ is set, but this solution requires a linear space $O(n)$ and, for n high, is absolutely not available on tiny devices.

Data Stream Analysis Streaming algorithms have received considerable attention in the recent past, due to the increasing gap between the rate of data generation and resources available to process them (see (102) for an overview). Estimating the number of distinct elements in a data stream is a primitive of fundamental importance in several fields where continuous and large data stream are implied. For example, in internet traffic monitoring, routers have usually a limited available memory, but it is extremely useful to collect various statistical properties, like the number of different destination addresses of the traffic flow for example, that could give indications on eventual on-going denial of service or port scan attacks (103, 104). It is immediately obvious how using the distinct counting primitive is of primary importance in this case, as the limited resources available and the (possibly) high traffic rate on routers would make it unfeasible to store every single element (being it an IP address or any other value of interest). The extension of streaming techniques to distributed settings has been motivated by applications in sensor networks (105, 106) , both assuming the presence of a reliable aggregation tree and in the more realistic scenario of multi-path routing protocols, where the partial information transmitted by each node is aggregated across different paths towards the base-station. Authors in (106) proposed a formal framework to study multi-path aggregation, called summary diffusion and they also gave formal necessary and sufficient conditions for the correct estimation of order insensitive statistics. (107) and (106) proposed multipath aggregation techniques based on the counting sketch by Flajolet and Martin (98). We emphasize that, differently from the above contributions, our emphasis is on networks of mobile agents.

4.2.1 Counting sketches techniques

In the previous section we described how the problem of the necessity of estimating the number of distinct elements arises in different fields of computer science and the reason why DISTINCT can be considered an extremely important computing primitive. Here we are interested in introducing the main algorithmic techniques proposed in literature to efficiently cope with this problem and to accurately approximate the DISTINCT value; a possible solution for approximating DISTINCT is given by the use of *sketches*, i.e. small summaries keeping trace of the number of distinct elements observed by each agent so far. It is important to remark that an exact solution of the duplicate data problem requires a linear amount of memory (this is a well-known theoretical result (108)) and another theoretical result again presented in (108) is that to obtain an ϵ-approximate solution, $\frac{1}{\epsilon^2}$ memory must be used; using logarithmic memory means that we will obtain only approximate solutions to the problem, as we will discuss in the

following. A very clear and useful classification of sketching techniques was made in (109), upon which we based the following descriptions.

Basic Flajolet-Martin Algorithm The first technique we present was introduced in the middle '80s by Flajolet and Martin (98), and most of the recent work is still based on it. The memory space used by FM-sketches consists of a bit vector M with size $L = (log(min(N, n))$, initialized to 0, where N is the number of items in the set of interest and n is the number of distinct elements. The main idea behind this technique is to assign each item of the data set (or data stream) to a random bit of M and set it to 1 with (quasi)-geometric distribution, i.e. $M[i]$ is selected with probability $\approx (2^{i+1})$. For the random selection of the bit in M an hash function h is used, that maps each value x uniformly at random to an integer in $[0, 2^{L-1}]$; in other words every time a value x appears in the stream, its hashed value $h(x)$ is computed, and it is converted in binary form; then on this binary string, the higher value y such that all the bits at the right of y are 0 is calculated, and finally the y-th position of the bit vector M is set to 1. The idea behind the algorithm is that using an hash function ensures that all items with the same value will make the same selection of bit in M, thus the final bit vector will be independent of any duplicates. In addition, for each distinct value, the bit b is selected with probability 2^{b+1} and thus we expect $M[b]$ to be set if there are at least 2^{b+1} distinct values and at most 2^b, since b is not set. The detailed mathematical analysis of the authors concludes to the point that the expected value K of distinct elements is close to $E[K] \approx log_2(0.77351 * n)$,so that $2^K/0.77351$ is a justified choice in that range. To reduce the variance in the estimator, the authors compute the average over tens of applications of the aforementioned procedure with a different hash function each time.

Probabilistic Counting Stochastic Averaging (PCSA) This second algorithm is a slight variation of the first proposal, presented by the same authors. Its main contribution is the reduction of the number of operations (per-item), at the cost of an increased standard error. The basic idea is to take the hashed values of the items only once and with each hashed input only the corresponding bit vector is updated. That is, if u is the value of an item, then the bit in position $h(u)/k$ of bitmap vector with address $a = h(u)mod(k)$ is set to 1. Then to compute the estimate, PCSA averages over the position of the least significant $0 - bits$, computes 2 to the power of that average and divides it by the correction factor 0.77351. At the end, the estimate is multiplied by k because of the fact that each bit vector has seen only $1/k$-th of the distinct items on average, according to the uniformly random distribution of the hashed values. As the hashing occurres only once for each item, the algorithm performs $O(1)$ operations on $log_2(n)$bit memory words for each item. The standard error which can be obtained using this solution is $0.78/\sqrt{k}$, where k is the number of bitmaps used. PCSA improves the per-item processing time from $O(k)$ of the basic algorithm (where k is the number

of bit vectors, each using a different hash function) to $O(1)$ with only a single hash function of the PCSA, on $log_2(n)$bit memory words.

LogLog Counting Algorithm The *LogLog Counting Algorithm* (presented and analyzed in (110) is actually a variant of the PCSA algorithm. The main contribution of this solution is that it reduces the size of the accumulation synopsis from $log(n)$ to $loglog(n)$. However, its drawback is that the standard error is increased from $0.78/\sqrt{k}$ to $1.30/\sqrt{k}$, where k is the number of bitmaps. This means that LogLog counting is less accurate, but the space complexity is improved logarithmically. The algorithm differs from the PCSA in two aspects:

- the position of the maximum bit set to 1 is maintained, in opposition to the PCSA, where the position of the least significant 0-bit is maintained;
- the overall function used to compute the estimate is different.

Time Decaying Sketches Algorithm This algorithm was introduced in (111) and is a complex approach to network data summarization. The new sketch maintains duplicate insensitivity, asynchronous arrivals and time decay of the processed data simultaneously. This approach has been specifically designed for sensor networks, while the two previous algorithms are general-purpose solutions. We are not interested in details on this algorithm, mainly because it has been shown in [35] that both time and space complexities are quite impossible to be obtained so as to conform to the theoretical results. Since we are looking for a light-weight, easy solution, we cited this algorithm only for completeness, but since we are interested only in the distinct counting problem, the time decaying sketches algorithm is not a suitable solution.

4.2.2 Our choice: details and motivations

After a detailed analysis of the different existing solutions, we decided to implement the first one (the basic FM sketch), as, according to the descriptions reported above, is the one giving the most accurate results. *LogLogCounting* algorithm has been discarded because, even if it reduces the space complexity, it introduces an higher error on the overall estimate, and the devices we consider in the application scenario we envisioned (as we will see in the next section) are equipped with enough memory to manage $O(log(n)$ bitmap vectors. The *Time Decaying Sketches* have been discarded because they are extremely complicated to implement in practice (essentially because they are capable of generating a lot more than distinct-values estimate, which is not necessary in our case for now), and because results in (109) pointed out that in some cases their processing time is extremely unsatisfactory. Another reason is that this approach is designed specifically for sensor networks, while the basic FM sketch is a general-purpose solution that better fits our proposal of a generic model of a mobile wireless network of tiny devices. We will present later in this chapter two different scenarios where the use of sketching techniques can be very interesting.

4.2.3 Distributed Network Monitoring: Privacy-Preserving Environment Monitoring

So far, we have seen how small portable devices carried by users can be exploited to provide useful information through active cooperation. To guarantee privacy, the system should use the information on the device without tracing the user. The first application we present is a privacy-preserving distributed technique to estimate the number of distinct mobile users in an area (we presented in (112)), a crucial information in critical situations like crowded airports or concerts where monitoring the number of persons could help organizing emergency countermeasures. In our scenario users periodically transmit a sketch (cfr.4.2.1) of the users they met in the past obtained applying suitable, duplicate-insensitive hash functions that also provide some kind of privacy, as it is not possible to retrieve the ID of the user from the hashed value. We tested our solution with simulations and we also implemented it on real hardware as a proof-of-concept. The main idea is to generate a map of the utilization of an area over time by a set of mobile users without disclosing their sensitive information. The only assumption we made is that each participant is identified by a unique id. When two users meet, they exchange their sketches (see figure 3.7a), to allow the involved nodes to estimate the number of distinct nodes (DISTINCT) they have met so far. This information is then occasionally communicated to a central server through a collector (see figure 3.7b). The Server displays the number of users in an area over time, and can be used by decision makers in architecture and urban planning or to calibrate/validate predictive models about pedestrians behaviors. We stress that while this technique has been previously used in (98, 102, 113), to the best of our knowledge this is the first study that considers mobile environments. The proposed technique has been designed to run on a network of resource constrained mobile devices, such as smart phones and/or wireless sensor nodes; in this context, the limited resources available at the devices require the adoption of optimized solutions both in terms of complexity and amount of exchanged data. These requirements make the trivial approach based on bit-masks unfeasible, since it requires an amount of memory that is linear in the size of the monitored population. Moreover, the technique we propose entails a degree of privacy, since the counting task is performed locally by each involved entity and only on the basis of the aggregate information exchanged among nodes. This implies that the central server only knows DISTINCT in the area, but not their identity and only a limited amount of memory (logarithmic in the size of the population) is required to store the relevant information.

Problem and model The basic problem is estimating DISTINCT in the network. Our goal is to make this process faster by exploting the fact that nodes moving in the network occasionally come close enough to exchange information. The idea is having each node maintain an estimate of DISTINCT encountered so far. We assume the presence of a special, fixed collector node, which receives data from any agent

occasionally moving in its vicinity and forwards them to a central server. We consider a set V of n distinct nodes, each equipped with a sensing device tagged with a unique ID within the network. After initial deployment, at time 0 nodes start moving according to the random way-point model (114) in the area. At any time t, only a subset of the node pairs can communicate, modeled by an undirected graph $G = (V, E_t)$, where the V is fixed and $(u, v) \in E_t$ if and only if u and v can communicate during round t. This model is pretty general and can account for multiple aspects, such as communication range and collisions in wireless networks. We assume that $(u, v) \in E_t$ if and only if their distance in round t is at most a given radius R. We assume that in a round t of communication a node can only broadcast a message to the set of its immediate neighbors, as defined by E_t

Each node in the network meets other nodes (possibly multiple times) over time and keeps a *sketch*, as introduced before. Considered a specific node u, we denote by Sk^u the sketch maintained by u. At a high level, each node u performs the following actions: i) when u meets another node v, it receives Sk^v and updates Sk^u accordingly; ii) after updating Sk^u, u broadcasts it to all nodes v, such that $(u, v) \in E_t$. To limit the amount of messages in the network, in step ii) a node broadcasts its Sk^u only if the new estimation of DISTINCT exceeds the previous one by more than a given threshold TH . We consider sketches that are *composable* and *duplicate insensitive* (115, 116, 117). Consider an order insensitive statistic of interest (i.e., whose value does not depend on the order in which events are observed), like DISTINCT we are considereing. A sketch Sk is *composable* if the following holds: let $Sk^u(S_1)$ be the sketch maintained by u at time t if it so far met the subset S_1 of nodes in the network. Assume u receives sketch $Sk^v(S_2)$ from v and let $merge(Sk^u(S_1), Sk^v(S_2))$, be the sketch resulting from the aggregation of $Sk^u(S_1)$ and $Sk^v(S_2)$, where $merge(\cdot)$ is a suitable sketch aggregation function depending on the statistic of interest and the sketching algorithm used to maintain it. Sk is composable if it is always the case that $Sk^u(S_1 \cup S_2) = merge(Sk^u(S_1), Sk^v(S_2))$. I.e., the sketch obtained by u after aggregating the sketch received from u is the same as the one u would have computed, had it encountered the whole set $S_1 \cup S_2$. In the remainder, we consider the sketches studied by Flajolet and Martin (98) and we use the phrase "FM sketch"to refer to any implementation of the original counting sketch. The use of composable and duplicate insensitive sketches has been considered previously for restricted and static distributed settings; in this work we are considering a dynamic network of moving wireless devices. FM sketches use a simple bitmap-based approach: every node ID (regarded without loss of generality as an integer value) is hashed onto a bit of a bitmap of length $k = O(\log M)$ bits, where M is an upper bound on the size of the universe (the number of nodes in the network in our case). In practice, $M = 2^k$, e.g., $k = 64$). The hash function $h(\cdot) : [n] \to \{0, \ldots, \log_2 M - 1\}$ used to this purpose is such that the probability of hashing onto the r-th bit is 2^{-r}. The bit to which the ID under consideration is hashed becomes 1 if it was not already. Considered any time t, let r denote the position of the least significant bit that is still 0 in the bitmap: it turns out that 2^r is a good estimator

for DISTINCT up to time t. To improve accuracy, we define a sketch as a collection of m bitmaps built as described above, each using an independently chosen hash function. If r_s denotes the least significant bit that is still 0 in the s-th bitmap, then $\frac{1}{m}\sum_{s=1}^{m} r_s$ is an accurate estimator of $\log_2 N_t$ if m is large enough (see theorem 1). The proposed sketch is composable and duplicate insensitive: considered two sketches $Sk^u(S_1)$ and $Sk^v(S_2)$, $Sk^u(S_1)\texttt{OR}Sk^v(S_2)$ is clearly the sketch corresponding to $S_1 \cup S_2$ and it is the same for both u and v. The scheme outlined above achieves excellent bounds in terms of resource efficiency and precision, as stated by the following

Theorem 1 ((98, 118, 119)) *Let $0 < \epsilon, \delta < 1$. At any point in time, every node u maintains an estimate $\hat{C}$ of the number N_t of distinct nodes encountered so far using $O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$ memory words, such that:*

$$\textbf{Prob}\Big[|\hat{C} - N_t| > \epsilon N_t\Big] \leq \delta.$$

Remarks. Some remarks are in order. First of all the naive strategy consisting in having nodes communicate their identities directly (and exclusively) to the sink when they eventually come close enough to it might require a long time till the sink has an accurate estimate. Furthermore, we assume for simplicity that the aggregate of interest has to be estimated at a special sink node, but the techniques we propose allow to address more general settings in which there is no distinguished sink. Finally, this approach can be immediately extended to estimate other aggregates using the same communication paradigm, for example sum or average (see (113)).

Experiments and Simulation Scenario In this section we describe the experiments we made to evaluate our algorithm. In our setting a set of users is inside an area of given size moving according to the random way point model and we want to estimate DISTINCT value without central coordination (the "collector" is never involved in any computation; it simply collects data). In every round of communication each node of the network performs the following actions:
1. if a message is received, the node merges its local_sketch with the sketch it just received from its neighbors
2. the node updates its estimate of DISTINCT
3. if the updated value exceeds the most recently propagated value by more than TH the current sketch is transmitted We consider a set of N distinct users; after the initial deployment, at time T users start moving randomly. The simulation experiments have been performed using the Shawn simulator (72) and we ran 50 simulations for each set of parameters. In the following, NNODES means number of nodes, MAXMSG means total messages, ROUNDS means total rounds and TH means sending threshold. We changed the following parameters for our experiments:in the 25x25 metres world we used:
TH = $\{0, 10, 25, 40, 50, 60, 75, 90, 100\}$; NNODES $\{25, 50, 100, 200\}$;

ROUNDS $\{20, 50, 100\}$,
while for the 100x100 metres world we used:
TH $\{0, 10, 25, 40, 50, 60, 75, 90, 100\}$; NNODES $\{25, 50, 100, 200, 500, 1000\}$ and
ROUNDS $\{50, 100, 1000\}$.

We change the size of the area and NNODES to verify the behavior of the algorithm under different situations of density. ROUNDS is related to MAXMSG generated by each node: we assumed that in every round a node can send a message depending on the value of TH : the higher ROUNDS , the higher the possibility for nodes to communicate. TH is of the utmost importance as it limits MAXMSG sent by each node but also because it allows a node to send or not an update message. For example, when TH = 0 we expect the higher MAXMSG as each node transmits a message at every round, but we also expect the maximum accuracy on the estimate, as the information is continuosly refreshed. When the TH is increased, it is possible that node is never able to collect sufficient information to pass the TH and thus to send its updated information, as we recorded in some experiments on the 100x100 area. We measured 2 fundamental metrics: **Error**(in percentage, ERR% in the following) on the estimated DISTINCT compared to the real DISTINCT (e.g. considering a network of 200, if the estimated number is 190, we have an error of 3%); **Number of messages** managed in average by each node. Please note that, implicitly, the number of messages sent in average by each node gives us an estimate of the energy consumed.

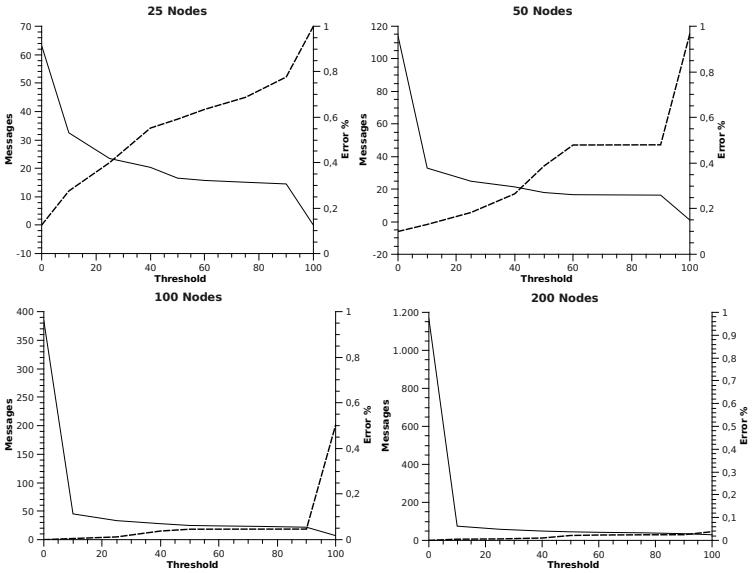
Results - World size 25x25 The first set of results refers to an area of 25x25 metres. **50 Rounds**. In figure 4.14a ROUNDS is 50, while TH and NNODES vary. On X axis we reported the different values of TH we used; on left Y axis we reported MAXMSG while on right Y axis, we reported ERR% (calculated as $1 - accuracy$, where $accuracy$ is defined as DISTINCT / NNODES). The full lines thus represents how MAXMSG varies with TH , while the dotted line represents how ERR% varies with TH . As can be observed in figure 4.14, ERR% is high for low network densities (25, 50 users), while it is lower as the density increases (100, 200). This is an expected behavior, and it is due to the fact that our algorithm spreads the information between nodes; with an higher density the information is diffused faster and to a greater number of nodes per round. The second aspect to underline is the importance of TH . A low TH (e.g. 0) means that in every round each node sends an update messages, then ERR% is reduced, since the information is continuosly refreshed but MAXMSG is extremely high. Having a look in particular at the 100 and 200 nodes scenarios, it is evident how even a very small TH value, like 10 reduces MAXMSG by a factor of about 10. Increasing TH has a significant impact on ERR% when the density is low, while it doesn't seem to disrupt too much the values obtained in a dense network. Thus, we can infer that when the density of the network is sufficiently high the performance in terms of accuracy is extremely good. Putting together the two needs of minimizing MAXMSG by increasing TH , and limiting ERR% by reducing it, it is possible to identify on the graph the "optimal" value TH * when the 2 lines intersect. This means, for example, that TH * for the 50 users case is 60, while TH * for the 200 users case is the maximum, 100, as the two

lines doesn't intersect. In this case we can obtain an error of $\approx 5\%$ with an average of ≈ 80 messages.
100 rounds.We can immediately note how increasing ROUNDS has a positive effect on the accuracy in sparse networks: in this scenario, even the 25 and 50 users experiments are very accurate when TH is 0. On the other hand, MAXMSG increases in the 100 rounds 200 users scenarios, confirming the intuition that when the network is dense, TH must be kept high.

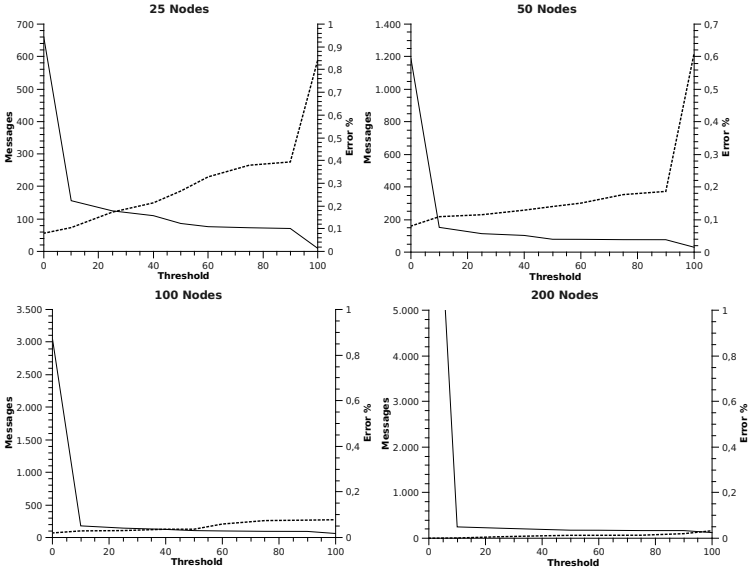
Results - World size 100x100 In the second set of experiments we increased the size of the area, thus reducing the density of users.
50 Rounds.The first figure refers to a monitoring of 50 rounds. As we can in figure 4.15, with a low number of users ERR% is constantly around 1, meaning that no user is able to communicate its sketch to others and this is due to the fact that the monitored area is so large that if the monitoring is performed for a very limited period, users almost never meet, and thus they cannot update their sketches. This intuition is confirmed by the fact that when the number of users increases $(500, 1000)$, with a very low TH we obtain an estimate, whose error is however high ($\approx 20\%$ for 1000 users and threshold 0), but at least some interactions between users happen.

1000 Rounds. With a very small TH (say 10 or even less), we are able to obtain a reasonable (ERR% below $\approx 30\%$) estimation for "sparse" networks $(25, 50, 100)$ managing less than 1000 messages for each user; if a greater accuracy is required, then a TH of 0 must be used, causing an average number of $\approx 1000 - 1500$ messages. It is interesting to notice how MAXMSG dramatically drops when a very small TH is used; this means that in a dense network the vast majority of messages carry redundant information, and thus can be avoided. Looking for example at the 1000 users case, it's easy to note that when TH is 0, MAXMSG is out of scale (it is around ≈ 26000), while when a TH of 10 is adopted, this number drops to less than 1000 messages with ERR% increasing just to $\approx 4\%$.t In conclusion, we can say that for sparse networks, a longer monitoring period and a low TH are needed to obtain a low ERR% ; as the density increases, to keep MAXMSG low, the sending threshold must be increased, but this doesn't impact on the accuracy of the estimation, as we have shown in both the area sizes.

Experiments on real hardware - proof of concept To assess the feasibility of our approach in practice, we implemented the algorithm on TelosB sensor nodes (120) running TinyOS (79).We used 30 static nodes from Motelab testbed (61). The main difference between simulations and real implementation is in how sketches are managed; since in TinyOS we must also cope with strong limitations on the size of messages, we decided to proceed as follows: each node stores some (20 in our implementation) uint32_t variables representing sketches; when sending a message these values are transmitted and converted in binary form by the receiver; then all the operations described before are performed and finally the resulting sketches are converted back in decimal



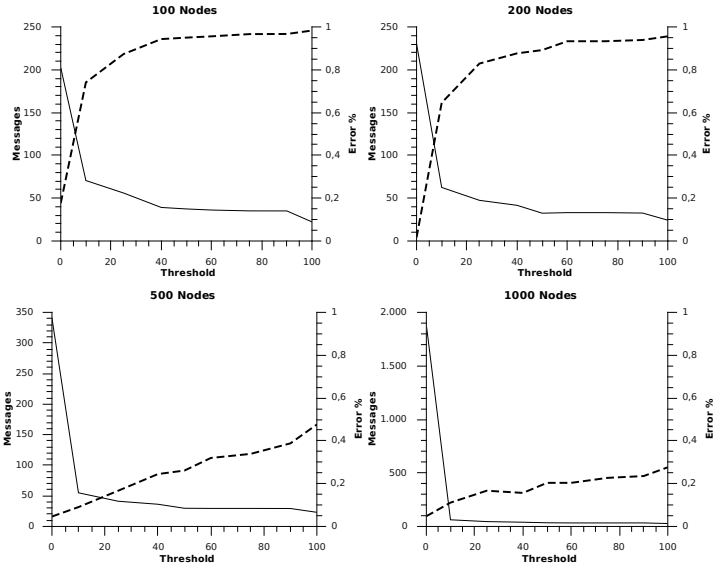
(a) 50 Rounds



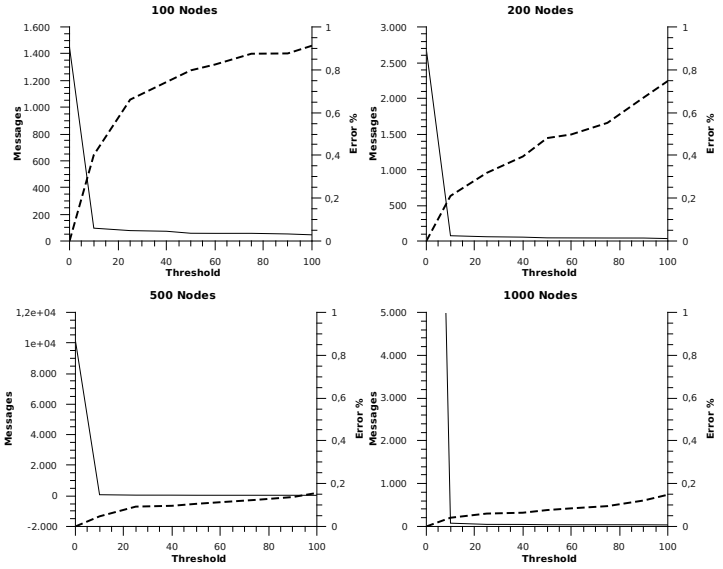
(b) 100 Rounds

Figure 4.14: Area size 25x25

form for the next transmission. The procedure is completely transparent to the final user and the rest of the algorithm is the same. With this assumption, we can build up a packet consisting of 82 bytes at the application layer (4 bytes * 20 sketches, plus 2 bytes of node ID), plus 17 bytes of header. The maximum size of a TinyOS packet is 128 bytes, thus our 99 bytes packets can be transmitted with no problems. We will



(a) 100 Rounds



(b) 1000 Rounds

Figure 4.15: Area size 100x100

not present detailed results on this scenario, as this is just a proof of concept; anyways we remark that: i)the error we recorded is compatible with the one we obtained from the simulation: in a scenario like the one we tested, using 30 nodes we obtained an estimate of 33 nodes with a TH of 10, thus the error was 10%; ii) the binary image is $\approx$ 36 Kb, where the programmable memory of Telosb nodes is 48 Kb, thus the memory is sufficient for our algorithm; iii) the computational power of the devices is sufficient to perform the operations required by the algorithm: when receiving a packet a node

is able to perform all the required operations in a few milliseconds.

4.3 Distributed Collaborative Filtering: Recommendations via SMS

We have described so far our experiments to evaluate the behavior of theoretical models in realistic conditions. As we remarked many times, the focus of this thesis is on dynamic mobile social networks, so next step is to identify one of the basic services offered by standard social networks and to implement it in an *Opportunistic* and *Efficient* way. For this reason, we thought of realizing a system of friendship recommendation in the social network of mobile phone users ((121) and extended journal version to appear in (122)). The application we realized does not assume any centralized coordination: it transparently collects and processes user information that is accessible in any mobile phone, such as the log of calls, the list of contacts or the inbox/outbox of short messages and exchanges it with other users. This information is used to recommend new friendships to other users. The information needed to perform recommendation is collected and exchanged between users in a privacy preserving way. Finally, information necessary to implement the application is exchanged transparently and opportunistically, by using the residual space in standard short messages occasionally exchanged between users. As a consequence, users are not required to change their habits in using SMS. The motivation why we decided to focus on SMS is that while western countries are experiencing the increasing availability of high speed connections and the diffusion of last generation smart phones with advanced interfaces to access mobile social networks, many still consider Short Messages the most convenient means for instant message exchange 1 . In any case, SMS traffic is still a consistent part of non-voice traffic. According to Lloyds (123), overall Person-to-Person SMS traffic has been 4.5 trillion of messages in 2008. These figures seem to justify the investments of some companies in social networking applications based on Short Messages, such as Jyngle2 [1] (124) and Peekamo (125). Furthermore, in large parts of the world, in particular Asia and Africa, SMS are expected to remain the primary means for data communication, at least in the near future. In 2007, nearly 1.5 trillion mobile messages were sent in the Asia-Pacific region (126).

4.3.1 Social Networking over SMS messaging

In our application, we consider that a node in the social network of mobile phone users is a mobile phone subscriber generating some amount of user-to-user communication. A link connecting two nodes represents an ongoing social relationship (e.g. nodes are friends, colleagues, classmates, etc.) between the corresponding users. In our approach, this social relationship can only be inferred estimating the users' *social profiles*

[1] Jyngle closed in August 2009.

similarity. Speaking in general terms, two users are similar when their social profiles are similar. In fact, the profile of a user is a general notion that depends on the information available to the system. In some cases this includes some biograhpical data, such as date of birth, sex, information about tastes, interests or activities. A profile is also completed by information that can be extracted transparently from the system, without explicit user intervention, such as the *log of calls*, the *list of contacts* or the *inbox/outbox of short messages*. We stress that in many cases, even limited information, e.g., the address book or the log of calls, can be used to infer possible relationships: for example, two users appearing in each other's address books are likely to be socially related, be it through a shared interest, a professional relationship, or simply because they are friends. Mining the social network underlying telephone traffic has been considered in the past, for example in (127, 128). Here, there is a (possibly labelled) link from A to B if A calls B at some point. The main goal in (127, 128) was to study the way in which such networks evolve over time, so as to infer and analyze probabilistic generative models (129) describing their evolution.

4.3.2 Recommending Social Relationships

Recommending new social relationships is one of the most basic services provided by social network applications. In our context, we are interested in strategies to recommend new contacts of potential interest to users. The challenge here is clearly to find contacts that are likely to share some common traits or, put differently, that are in some way "similar" to the user to whom the recommendation is being provided. As stated, this problem is very close to the link prediction problem studied by Liben-Nowell and Kleinberg (130), whose focus is on statistical indicators of social closeness and not on their efficient and decentralized computation. More formally, if a node A recommends a node B to a third node C, A is suggesting a potential interest or utility for C in establishing a contact with B (unless this contact already exists). Recommendation is performed on the basis of knowledge about the social profiles $L(B)$ and $L(C)$, which are used to estimate the extent to which B and C are "similar". The underlying assumption, made more precise in the following, is that the more similar B and C, the more likely it is that they either have a contact, or they might benefit from establishing one.

Privacy requirements make the explicit exchange of private profile information or user contact lists unrealistic for applications. Furthermore, data must fit into the residual space of person-to-person short messages and thus they must be represented in a compact form (i.e. a *sketch*). Figure 4.16 outlines the general application scenario we consider. In step 1, users A and B compute the sketches $sk(L(A))$ and $sk(L(B))$ of their respective social profiles. As observed before, this is a compact representation of the user's social profile preserving her privacy. In step 2 and 3, A and B occasionally send a short message to C. The message space is partially filled with some personal text (e.g. SM Text = "shall we meet this evening?") while the residual space is exploited

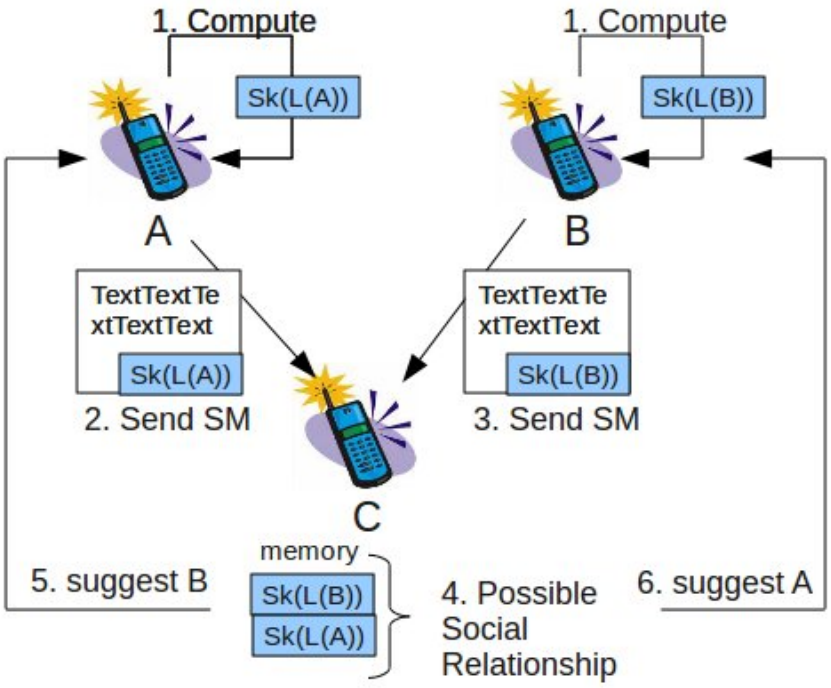


Figure 4.16: A scenario

to deliver the sketches. Observe that users interact with the SMS as usual, while the residual space is transparently managed by a suitable application. In step 4, user C (i.e. the recommender) infer a high degree of similarity between A and B on the basis of their respective sketches. In steps 5 and 6, C eventually recommends a possible friendship to users A and B.

4.3.3 Locally inferring community structure

One of the main issues in recommendation systems for social networking is predicting the potential benefit of new links between users. In the fully decentralized scenario we consider here, this amounts to answering the following question: when should a user A recommend a contact between two other users B and C she is aware of. This in turn implies a number of other issues: i) What information about B and C does A combine in order to decide whether or not she should suggest a contact between B and C if not existing already; ii) how is this information obtained, manipulated and exchanged; iii) how are computational, storage and communication constraints met; iv) how is privacy preserved.

Alike many networking applications, we recommend new contacts on the basis of similarities between users. Thus, A will recommend B and C to establish a contact if A assesses that B and C are “similar”.

In particular, if we view profiles as feature sets, we say that two users A and B are *similar* when their social profiles $L(A)$ and $L(B)$ overlap significantly. In this perspective, we estimate user similarity by the Jaccard coefficient $J(L(A), L(B)) = \frac{|L(A) \cap L(B)|}{|L(A) \cup L(B)|}$, a widely accpeted measure of similarity between sets. In the social networking scenario we consider, it captures the well known fact (129, 130) that social networks are densely connected at a local level or, roughly put, the folklore that two friends of the same person are significantly more likely to be friends than any two randomly chosen people.

4.3.4 Estimation of the Jaccard coefficient.

Consider the set of possible contact identifiers. Recall that, as motivated further in this section, they can be regarded as integer numbers falling in the range $[n] = \{0, \ldots, n-1\}$ for suitable n. The only assumption we need is that they are unique, a constraint that is met in practice in the applications we consider; they are users' telephone numbers or a suitable representation of them. As a consequence, considered any two users A and B, their contact lists $L(A)$ and $L(B)$ may be simply regarded as two subsets of $[n]$. Our goal is to measure their overlap using the Jaccard coefficient: $J(L(A), L(B)) = \frac{|L(A) \cap L(B)|}{|L(A) \cup L(B)|}$.

A very simple and elegant technique to estimate the Jaccard coefficient has been proposed in several equivalent forms by Broder et al. (131, 132). Assume we are able to choose a permutation $\pi(\cdot)$ mapping $[n]$ onto itself uniformly at random. For every $X \subseteq [n]$, denote by $\pi(X)$ the set of the images of elements in X when $\pi(\cdot)$ is applied and let $\min(\pi(X))$ denote their minimum. Then it can be shown (131) that (i) considered a set $S \subseteq [n]$ and for every $a \in S$, $\mathbf{Prob}[a = \arg\min(\pi(S))] = 1/|S|$; (ii) for every $S_1, S_2 \subseteq [n]$: $\mathbf{Prob}[\min(\pi(S_1)) = \min(\pi(S_2))] = J(S_1, S_2)$. This property immediately yields a technique to estimate $J(S_1, S_2)$.

The algorithm consists in performing m independent executions of the following procedure: i) pick one permutation $\pi(\cdot)$ of [n] uniformly at random from the $n!$ possible ones; ii) in the i-th iteration, let $\min(S_1) = \min(\pi(S_1))$ and $\min(S_2) = \min(\pi(S_2))$. We increment a counter C_m whenever $\min(S_1) = \min(S_2)$. At the end of the process, our estimation of $J(S_1, S_2)$ is C_m/m. Standard tools from probability theory tell us that C_m is an increasingly (with m) accurate estimation of $J(S_1, S_2)$.

Computing and maintaining contact list sketches. Unfortunately, generating permutations uniformly at random requires a number of truly random bits that is in the order of n (131). Fortunately, suitable families of simple, linear hash functions perform well in practice (e.g. see (133, 134)). In particular, we use linear permutations (134), i.e., functions of the form $h(x) = ((ax + b) \mod p) \mod n$. Here, p is large prime, while a and b are integers belonging to the intervals $[1, p-1]$ and $[0, p-1]$ respectively.

We next describe how each node A of the network maintains the local *sketch* $sk(A)$ associated to $L(A)$. As pointed out before, we assume below that every number in $L(A)$ is an integer falling in $[n]$. To this purpose, it is enough to perform a first step in which each contact identifier (e.g., a user's mobile phone number) is regarded as a string and

```
UPDATE(sk(A), pn)
Require: Sketch sk(A), number pn
 1: x = hash(pn) {Hash pn to an integer in [n]}
 2: for i: 1 ...m do
 3:   M_i = h_i(x) {Map x according to a random permutation}
 4:   if M_i < min_i(A) then
 5:     min_i(A) = M_i
 6:   end if
 7: end for
 8: return sk(A)
```

Figure 4.17: Update algorithm.

this string is mapped onto an integer in $[n]$, using any hash function, as long as the probability of collision is sufficiently small. This is for instance the case if we hash contact identifiers to 32-bit integers using a good hash function, e.g., implemented in Java standard classes. As a second step, m hash functions are generated. The i-th hash function has the form $h_i(x) = ((a_i x + b_i) \mod p) \mod n$. The integers $\{a_1, b_1, \ldots, a_m, b_m\}$ are generated *independently* and uniformly at random, respectively in the interval $[1, p-1]$ for the a_i's and $[0, p-1]$ for the b_i's. Finally, for $i = 1, \ldots, m$, let $\min_i(A) = \min_{x \in L(A)}\{h_i(x)\}$. The sketch of $L(A)$ is the *ordered* vector $sk(A) = (\min_1(A), \ldots, \min_m(A))$. A version of this algorithm that allows dynamic updates when new numbers are added to the contact list is given in Figure 4.17.

The cost of algorithm `UPDATE(`$sk(A)$`, pn)` is $O(m)$. The deletion of items from the contact list is more expensive, since the element removed might be the one achieving minimum value on one or more of the hash functions. Therefore, in the case of deletions $sk(A)$ has to be recomputed from scratch and the cost becomes $O(m|L(A)|)$. Note however, that m is in the order of a few tenths at most (10 in our experiments). This complexity is therefore fully compatible with standard commercial mobile phones.

In addition to $sk(A)$, A's device stores $sk(B)$, if available, for every B in her contact list. The required amount of additional memory, as discussed further in greater detail, is a few tenths of bytes for each entry in the contact list (40 in the current implementation), thus perfectly compatible with standard commercial devices.

Exchanging sketches. In the scenario we envision, if both user A and B run the application and B sends an SMS to A, B will use the available free space of the message to send its own sketch $sk(B)$, or part of it, to A. Let's assume for the moment that there is enough residual space in the message to send the whole $sk(B)$. Note that this is likely to be often the case since, as we see later, the size of a sketch is typically a few tenths of bytes, 40 in the present implementation. Morover, we discuss how to

RECOMMEND(A, $sk(B)$, $sk(C)$, θ)

Require: Node A, Sketch $sk(A)$, $sk(C)$, threshold θ

1: Estimate $J(L(B), L(C))$ from $sk(B)$ and $sk(C)$ {Node must have both}
2: **if** $J(L(B), L(C)) > \theta$ **then**
3: A recommends B to C or viceversa
4: **end if**

Figure 4.18: Recommendation algorithm.

address cases in which the SMS free space is not sufficient to contain $sk(B)$ in a further paragraph of this section. Whenever A's device receives the message, it transparently extracts $sk(B)$ from the message body. If B is one of A's contacts, then $sk(B)$ is stored in A's contact list, associated to B, possibly replacing an older copy of $sk(B)$.

Fully decentralized recommendation of contacts. Recall that we assume that two users are *similar* to the purpose of the application whenever their contact lists overlap significantly. The algorithm in Figure 4.18 implements this general idea. In particular, the algorithm describes the behaviour of the generic, mobile terminal of some user A. If A has the sketches of both B's and C's contact lists, A will recommend B (C) to C (B) whenever the local estimation of $J(L(B), L(C))$ exceeds some given threshold θ. In Section 4.3.7 we study, among others, how the choice of the threshold affects the quality of recommended contacts.

4.3.5 Implementation issues

Memory requirements. If we consider the generic node A, the amount of memory needed to store its contact list is $\Theta(L(A))$. In our implementation, A also needs to store i) its own sketch $sk(A)$ and, in the worst case, ii) $sk(B)$, for a subset of nodes from which A received SMS messages in the past. If we assume that A stores the sketch of every contact, the required amount of memory is $O(m(|L(A)|))$. In practice, if we use $m = 10$, the additive amount of bytes required for each contact is about 40. This is in the same order of magnitude of an entry in any address book of a commercial device.
Computational costs. The computational cost of maintaining sketches and providing recommendations is also compatible with current commercial devices. In particular, adding a new contact to the contact list of a node A requires updating $sk(A)$ (algorithm UPDATE$(\cdots)$ in Figure 4.17) and has cost $(O(m))$. For two nodes B and C other than A, deciding at A as to whether recommending each of them to the other requires estimating $J(L(B), L(C))$, which has cost $O(m)$. Finally, removing a contact from $L(A)$ (typically a far less frequent operation) is more expensive but it has (up to m) still linear cost, i.e., $O(m|L(A)|)$. As to this point, it should be noted that in the social networking applications we envision, $|L(A)|$ is closely related to Dunbar's number, i.e.,

the maximum number of contacts a human can manage (135) in a social network. This number is between 150 and 200 in a physical social network and it is only slightly larger in online social networking platforms, with a very small fraction of users exceeding a few hundred contacts in Facebook. To summarize, the cost of removing a contact i) is typically an infrequent operation, ii) it has linear cost with respect to the size of the contact list and iii) this size is relatively small in practice. Computation is performed at user devices. Nowadays, these are typically small computers, whose capabilities are perfectly compatible with the computational effort required by the proposed techniques.

Hash functions. The number of hash functions required (i.e., m) is chosen, so that probability that the estimation of the Jaccard coefficient differs from the true value by more than a chosen constant is below a suitably small constant. We refer the reader to specific work (e.g., (131, 132)) for technical details. In our case, experimental evidence suggests that already 10 hash functions are sufficient to strike a reasonable balance between accuracy of the estimation and memory requirements. A further constraint is that all user devices use the same set of hash functions. In practice, hash functions and the algorithms we propose will be implemented and maintained in the device's memory. This in turn requires storing, for each hash function, its coefficients and p in binary form. In our implementation, coefficients are 32-bit integers, while p is the well-known Mersenne prime $2^{31} - 1$, which does not need to be stored explicitely. So, it turns out that the actual storage requirements for maintaining hash functions is around 80 bytes. The overall implementation (code, hash functions, runtime data structures) requires less than 1Kbyte space. To this, we must add the (variable) size of the user's (modified) contact list. Thus, the storage requirement of the modified contact list is in the same order of magnitude as in a standard implementation.

Sketch size vs message body size. We observed earlier that we cannot always assume that the message body of an SMS sent from some node A to another node B has enough free space to host $sk(A)$. The most direct way to circumvent this problem is for A to send its sketch whenever the avaiable free space in the message body exceeds $|sk(A)|$. In fact, the distribution of SMS message sizes seems to be approximately uniform (136). Assuming for the sake of simplicity that it is exactly uniform and that message sizes of different messages are independent variables, we have that half of the messages have 80 bytes available space in the average, more than 75% have at least 40 bytes available to carry sketches and so on. This means that, in the average, 1.34 message are enough for A to send its sketch to B, which means that, in practice, if A sends 2 SMS to B, the latter is very likely to receive A's sketch. [1]

Choosing the right threshold. As we show in Section 4.3.7, the predictive accuracy of the heuristics we consider is sensitive to the choice of the similarity threshold. Also, it turns out that the right choice for the threshold can depend on the social network

[1] An alternative solution is that A sends to B part of its sketch, compatibly with the available space in the SMS message body. This solution requires bookeeping both at A and B, to keep track of the portions of $sk(A)$ still missing at B. In fact, the former solution can be more easily implemented than the latter and it requires no additional data structures.

under consideration. Choosing the right threshold in a practical application is an issue. Though the purpose of this paper is showing that successfully recommending links is possible in a fully decentralized way, we briefly outline below two approaches that might be used in practice to address this issue.

The first approach is centralized in nature. It consists in using a small sample of the population (as, for example, the one from the Reality Mining Project) to estimate the correct threshold value. Assuming that we are able to access a small sample of the population can be reasonable in most cases.

To address cases in which this were not possible, we also consider a second approach that is fully decentralized and is based on learning via explicit feedback. In particular, if user A recommends a new contact C to user B, B can provide positive or negative feedback to A about the relevance of the suggestion received (e.g., in the form of a binary relevant/non-relevant feedback). This can require explicit notification from user B or it can be done transparently. For example, A might receive a positive feedback from user B (e.g., contained in a special SMS) whenever C is added to B's contact list. In this way, A can over time locally compute precision and recall values for the set of recommendations it provided, adjusting the value of threshold accordingly. In particular, since precision (recall) increases (decreases) as the threshold increases, a binary search allows to find a threshold value that strikes a good balance between precision and recall using a moderate number of feedbacks. It should be noted that, in this case, different users would in general use different values for the threshold (though we expect them to be close), but this does not affect in any way the implementation of the heuristics, which remains unchanged.

4.3.6 Enforcing privacy

As described in the previous sections, a sketch is a representation of the contact list that, besides reducing the amount of data to be exchanged, does not fully discloses a user's contact list. As an example of the type of information that is leaked by the sketch $sk(A) = (\min_1(A), \ldots, \min_m(A))$ of contact list $L(A)$, we point out that if $h_i(x) < \min_i(A)$ then certainly $x \notin L(A)$. In this section we show how to securely compute the Jaccard coefficient of two contact lists, $L(A)$ and $L(B)$, without revealing any information except what can be deduced from the Jaccard coefficient itself.

Let us consider two parties A and B, each holding a vector of length m; with a slight abuse of notation we identify each party with his/her input vector. In our application to the computation of the Jaccard coefficient, the vectors will be the sketches of the respective contact lists. A and B wish to compute the number of positions i for which $A[i] = B[i]$ without revealing any additional information on the vectors. We will describe a protocol that uses an additively homomorphic encryption scheme $(E; D; K)$ like Paillier cryptosystem (see (137) for further information).

Homomorphic encryption scheme. Let $(E; D; K)$ be a homomorphic encryption scheme and assume that the message space for a public key pk returned by the key

generator algorithm K on input security parameter m is $\mathbb{Z}_p$ for some integer p of length m. The following additive homomorphic properties hold: i) the product of two ciphertexts is a ciphertext for the sum of the plaintexts; that is, for all messages $a; b \in \mathbb{Z}_p$ and public keys pk, we have $D(E(pk, a)\cdot E(pk, b), sk) = a + b$; ii) raising a ciphertext for message a to power r gives a ciphertext for $r\cdot a$; that is, for all $r \in \mathbb{Z}_p$ we have that $D(E(pk, a)^r, sk) = r\cdot a$.

The protocol. The protocol can be described as follows:

1. A picks a pair of public and secret key (pk, sk) for encryption scheme (E, D, K) by running the key generator algorithm K on input 1^m; for $i \in [n]$, A computes encryption $a_i = E(pk, A[i])$ of $A[i]$; A sends pk and $(a_i)_{i\in[n]}$ to B;
2. for $i \in [n]$, B computes encryption $b_i = E(pk, -B[i])$ of $-B[i]$, picks random $r_i \in \mathbb{Z}_p$ and sets $c_i = (a_i + b_i)^{r_i}$. Notice that by the homomorphic properties of (E, D, K), c_i is a ciphertext for $r_i\cdot (A[i] - B[i])$. Therefore if $A[i] = B[i]$, then c_i is an encryption of 0; otherwise c_i is an encryption of a random element of $\mathbb{Z}_p$. B randomly permutes the c_i's and sends them to A.
3. A decrypts the m ciphertexts received from A, counts the number s of ciphertexts that are an encryption of 0 and sends s to B.

Properties of the protocol. The protocol can be characterized from the following simple observations:
Correctness. The value s computed by the protocol is the number of indices i for which $A[i] = B[i]$, with probability exponentially close to 1.
Privacy of the input. Each of A and B gets no information on the other party's vector, besides what can be obtained from the output of the protocol. For A, this can be easily seen by exhibiting a probabilistic polynomial-time simulator S that, for all vectors A and B, on input vector A and the number s of positions in which A and B coincide (but not vector B) outputs A's view of the protocol. Similarly, we can construct a simulator for B.
Efficiency and accuracy. To obtain the standard level of security usually associated with public key encryption, we can use the Paillier cryptosystem with 1024 bit modulus. Therefore one encrypted sketch will be 256 byte long and thus, under the assumptions of Section 4.5, about 3 SMS are needed to send one encrypted sketch. We do observe though that this level of security might be an overkill for our application since we do not encrypt private data but we use a hash that partially (but not completely) hides the private data. Therefore, for our specific application, the length of the modulus can be reduced to 512 bits, thus halving the space requirement. We stress that breaking the Paillier cryptosystem with 512 bit modulus requires a few months of computation.

As to prediction accuracy, it should be noted that the approach we propose does not alter the sketches. As a consequence, the performance of the protocol with respect to accuracy is unaffected and the experimental results of Section 4.3.7, in particular as regards precision and recall, also hold for the encrypted version of our contact recommendation approach.

Coming back to the recommendation system, we have two parties A and B, each with a private contact list, $L(A)$ and $L(B)$, that wish to compute the Jaccard coefficient $J(L(A), L(B))$. Obviously, the Jaccard coefficient can be computed by applying the above protocol to the characteristic vector of the two sets. The protocol will then run in time linear in the size of the underlying universe set. A much more efficient protocol is instead obtained by running the above protocol with each party holding as an input the sketch of his/her contact list computed using the same sequence of random (or min-wise independent) permutations.

4.3.7 Experimental Analysis

In this section we present the results of experimental analysis on real, publicly available data sets, in our opinion supporting the effectiveness of the approach we propose.

Experimental setting Our experimental work had the following main goals. In the first place, we wanted to understand the intrinsic effectivess of the Jaccard coefficient to infer social relationships in the network of mobile phone users. A further issue was to assess whether the techniques we use to approximate the Jaccard coefficient are compatible with the hard space constraints, imposed by Short Message size. In particular, these severely limit the number of hash functions that can be used to compute contact list sketches, which in turn affects the accuracy of the estimation, especially when the value of the Jaccard coefficient is relatively small in absolute terms, as is the case for one of the datasets we consider. Finally, we wanted to investigate the effectiveness of our overall approach in suggesting contacts to users. The problem here is that the public datasets we used did not allow us to directly assess the *a posteriori* effect of recommendations on users' choices. For this reason, in our experiments we considered the ability of our approach to predict existing links as a proxy for its effectiveness in providing useful recommendations.

Data sets and contact graphs In this subsection, we describe the datasets we used and how we extracted from them the social networks we considered for the experiments. The first dataset contains data obtained from a sample of real mobile phone users over a relatively large time interval. As such, the dataset is ideal to test the effectiveness of the recommendation strategy we propose. On the other hand, the sample size is about 100 users among which a few thousand contacts (i.e., phone calls) were recorded. Furthermore, the sample might be statistically biased, since the involved people belong to the same university campus. For this reasons, we also performed our experiments on a far larger dataset. This refers to a different application (i.e., it is a sample of the social network of Facebook users), but some general structural properties of both networks (e.g., degree distribution and clustering coefficient) are similar and follow patterns largely observed in social networks, as further commented in the paragraphs that follow.

The Reality Mining Project dateset. Accessing telephone traffic data is far from trivial, since very few public datasets are available. The Reality Mining project (4, 44) represents the largest mobile phone experiment ever attempted in academia. Its dataset contains thousands hours of continuous data on daily human behavior and contains information on *call logs*, Bluetooth devices in proximity, cell tower IDs, application usage, phone status.

The Facebook dataset. In order to perform a more exhaustive and complete validation of the discussed techniques, we replicated the experiments on a much larger, real dataset obtained from a crawl of the Facebook social network (138)[1]. This dataset is the result of a sampling based on the Metropolis-Hastings Random Walk (MHRW)(139, 140) over the graph of Facebook users. This technique yields a uniform stationary distribution of nodes (users), resulting in a dataset with 957K unique nodes and their relationships.

Contact graphs. We extracted a *contact graph*, i.e., a graph describing contacts among users of the two applications, from the two datasets we considered. For the Reality Mining project dataset, we used call logs to build a *contact graph* where nodes are mobile users characterized by unique ids (i.e., telephone numbers) and there is an edge connecting two users i and j if and only if the call log contains at least 1 call occurring between i and j. Formally, in this case the contact graph $G(V, E)$ was obtained as follows:

- V is the set of users appearing in the log of calls
- for each pair$(i, j) \in V$, edge $(i, j) \in E$ if and only if at least 1 call occurred between i and j.

To avoid problems related to data incompleteness, we restricted our experiments only to the people actually participating in the Reality Mining project (around 100 people), whose logs are complete and accurate.

For the Facebook dataset, we directly used friendship relationships to construct a contact graph. In particular, there is an (undirected) link between nodes i and j whenever the corresponding users are friends in the Facebook social network.

Note that the contact graphs for both cases are *undirected*. This is perfectly consistent with Facebook's dataset, in which contacts represent friendships in the social network. For the Reality Mining Project dataset, this entails assuming that contact lists are *symmetric*, i.e., i belongs to the contact list of j (and viceversa) if at least 1 call occurred between i and j during the period of observation.

[1]The dataset is publicly available for research purposes at
$http://odysseas.calit2.uci.edu/doku.php/public:online\_social\_networks\#facebook\_social\_graph$.

Require: $G_w(V, E)$
1: **for** $v_1 \dots v_n \in V$ do
2: for each v_a, v_b among the contacts of v_i do
3: retrieve $sk(A)$, $sk(B)$ by emulating an SMS reception
4: RECOMMEND(A, $sk(A)$, $sk(B)$, θ)
5: end for
6: end for

Figure 4.19: Simulation algorithm.

Comparison between datasets. Even though the application scenarios for the two datasets we consider are different, some important statistical properties of the resulting social networks are similar. For example, the degree distributions for the two networks both follow power laws and their upper parts are roughly approximated by a Zipf law with parameter close to 1. Also, the clustering coefficient of the Facebook contact graph is 0.0021. This is the same order of magnitude as the Reality Mining Project contact graph (0.0095), even though significantly smaller. This may be because the former dataset refers to a statistically biased, relatively small sample of people who are more likely to know each other. Finally, as we show in Subsection 4.3.8, both datasets show the same, strong correlation between overlap in contacts lists and social ties. This similarity is not surprising, but it is rather a well known common trait to many social networks describing human interactions (129).
Remark. For the Reality Mining Project dataset it is possible to define different contact graphs, describing networks of increasingly strong social ties. In particular, it is possible to define a graph $G_w(V, E)$ in which we declare the existence of a link between i and j if and only if at least w calls occurred between them. In this way, we can filter out occasional contacts. Though this is potentially interesting, in our experiments we restricted to $G_1(V, E)$, since the Reality Mining Project dataset is relatively small and higher values of w further reduced its size.

Experimental scenario for recommendations We assessed the quality of our technique in providing recommendations of good quality by using its ability to uncover existing relationships as a proxy. In particular, for each node v_i and for each pair $\{v_a, v_b\}$ both belonging to v_i's contact list, we ran our algorithm to predict the existence or non-existence of link (v_a, v_b). We then checked whether the link existed or not. This is synthetically described by the algorithm in Figure 4.19, which simulates the general recommendation algorithm described in figure 4.18. As to user profiles over which sketches are computed, for each user v, we considered the set of v's neighbours as her social profile, namely her list of contacts. As remarked earlier, this is a minimalistic assumption and corresponds to information that can be transparently accessed on virtually any current commercial device.

Performance indices The error of the recommendation strategy we propose is potentially affected by two factors: i) inaccuracy in the estimation of the actual value of the Jaccard coefficient; ii) error in the recommendation itself, i.e., the contact we recommend is not interesting to the user. These two aspects are clearly interrelated in complex ways. We considered these two contributions separately, which corresponds to the worst-case assumptions that the effects of the two sources of error sum up.

Effect of accuracy in the estimation of the Jaccard coefficient. Assessing the accuracy of our approximation of the actual Jaccard coefficient poses some issues. In the first place, our data show that even values of the Jaccard coefficient related to a significant degree of social relationship can be low in absolute terms. This makes an accurate estimation harder to attain given the stringent constraints we have to comply with. In particular, if we use m hash functions, we only have m possible values for our estimation of the Jaccard coefficient. When $m = 10$ as we assume, this provides very little granularity. Namely, possible values of the estimated Jaccard coefficient are $0.1j$, with $j = 0, \ldots, 10$, whereas values of the true Jaccard coefficient corresponding to a significant degree of social interaction are around $[0.05, 0.1]$ in the Reality Mining Project dataset.

On the other hand, our algorithm is threshold-based: it recommends a contact between two nodes A and B whenever $J(L(A), L(B))$ is above a given threshold. For this reasons, we consider the *Jaccard-Estimation Performance* (JEP) index, defined as the fraction of times that our algorithm gives the same recommendation as it would give if it knew the exact values of the Jaccard coefficient. We call these two versions of the algorithm *apxJacc* and *exactJacc* in the paragraphs that follow. Formally, for every node i, let C_i denote the number of times that *apxJacc* and *exactJacc* take the same recommendation decision for pairs of nodes beloning to i's contact list.

The *Jaccard-Estimation Performance* (JEP) index is formally defined as:

$$JEP = \frac{\sum_{1 \leq i \leq |V|} C_i}{t}$$

where V is the vertex set, i.e., the overall number of users and t is the overall number of node pairs evaluated.

Quality of recommendations. As observed earlier, the datasets we used do not allow to test the a posteriori effect of recommendations. For this reason, we tested the ability of our approach to provide high quality recommendations to users as a proxy for its effectiveness. To this purpose, we checked to which extent the contacts that are recommended correspond to actual links, thus evaluating *precision* (i.e., the fraction of existing links that have been recommended over the total number of given recommendations) and *recall* (i.e., fraction of all existing links that have been recommended over the total number of actual links) of our recommendation algorithm.

4.3.8 Experimental results

In this section, we provide experimental results that address the following issues: i) whether or not the Jaccard coefficient is a good indicator of social ties in the datasets we consider and which are reasonable threshold values for the recommendation heuristic we propose; ii) how good is our estimation of the Jaccard coefficient, at least in the sense made precise in the previous subsection; iii) how good are the recommendations we provide, which we indirectly answer by the extent to which we are able to infer existing contacts between node pairs.

Heuristics In order to evaluate the validity of the Jaccard indicator to recommend social ties, we will also use the Adamic-Adar coefficient, introduced in (141) and recognized in (142) as one of the best indicators to evaluate the similarity between users. According to Adamic Adar coefficient, the similarity between two users in a set is measured as follows:

$$similarity(A,B) = \sum_{shared\_items} \frac{1}{log[frequency(shared\_item)]}$$

Where the frequency of an item is measured as the number of times the specific item is possessed/named by users. In the data sets we are actually considering, each time a given contact is in the phonebook (for reality mining dataset) or in the list of friends (for Facebook dataset) of a user, then the frequency of that item is increased by one. This metric weighs more rare items.

Thus, summing up, we will evaluate the following metrics:

- Approximation of Jaccard Coefficient, using our proposed approach
- Exact Jaccard Coefficient, to show the validity of our approximation
- Adamic Adar Coefficient, to confirm the validity of Jaccard coefficient as a similarity metric

The algorithm we used to estimate the Adamic Adar coefficient is the same we used for the Jaccard coefficient and described in figure 4.18; in other words a friendship is recommended if the similarity between the two users, according to Adamic Adar coefficient, is greater than a given threshold.

Experimental setting The algorithms we propose depend on the similarity threshold θ and on the number m of hash functions used to generate sketches of user profiles. Furthermore, they are are probabilistic in the generation of the hash functions used to estimate the Jaccard coefficient. For this reason, to generate the results described in the following Subsections 4.3.8.1 and 4.3.8.2, we performed 10 independent runs of the algorithms for every pair of (θ, m) we considered and we averaged the results over the 10 trials. In particular, in each run we generated m hash functions, independently

at random according to the guidelines given previously. We used Java Math's built-in pseudo-random generator to produce the coefficients of each hash function, each time initializing the pseudo-random generator object with a different seed to enforce independence.

As to the values of θ and m, we considered $\theta \in \{0.0, 0.01, 0.02, ..., 1.0\}$, while for the number of hash functions we took $m \in \{10, 20\}$ for the Reality Mining Project dataset and $m \in$\{10,20,40,60,80,100\} for the Facebook dataset. The reason for considering a larger range of values for m in the Facebook scenario was testing the effect of increasing the number of hash functions beyond values that are of practical interest in the scenario we envision but might be feasible in others.

Jaccard coefficient and social ties Figure 4.20 synthetically describes the correlation existing between values of the Jaccard coefficient and existence of links between node pairs for the Reality Mining Project dataset. More in detail, the x-axis is divided into intervals of width 0.05 each, starting at 0.0 and ending at 0.2. For the j-th interval ($j = 0, 1, 2, 3$), the ordinate represents the fraction of pairs (A, B) of users such that i) $J(L(A), L(B))$ falls in the interval $[0.05j, 0.05(j+1)]$ and ii) A and B are contacts, i.e., they are in each other's contact lists. The x-interval stops at the value 0.2, since we observed too few pairs with Jaccard coefficient beyond this interval, to be statistically meaningful. This picture clearly shows that the Jaccard coefficient is a good indicator of social ties in mobile user networks. Furthermore, at least in the datasets we considered, the Jaccard coefficient allows to identify a sharp transition around the value 0.05, from a region characterized by sporadic ties to one characterized by frequent social relationships. In light of these observations, we chose the value 0.05 as a threshold in our recommendation algorithm.

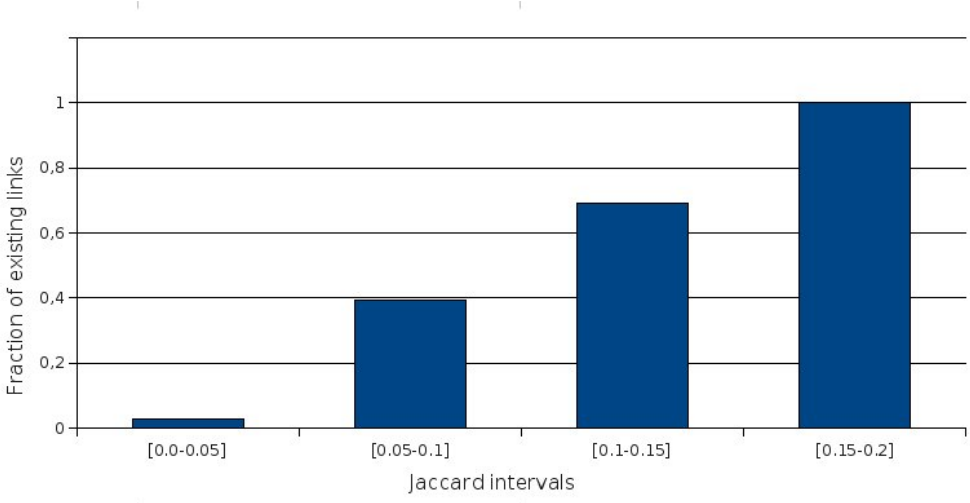


Figure 4.20: Existing linked pairs over total pairs, Jaccard coefficient

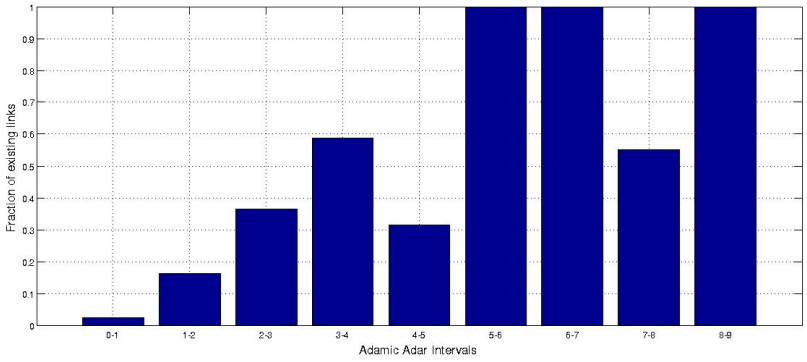


Figure 4.21: Existing linked pairs over total pairs, Adamic Adar coefficient

Figure 4.22 reports similar results for the Facebook dataset. The x and y-axes have the same meaning as before, with the only difference that, given the much larger size of the dataset, in this case we have a statistically significant number of pairs with very high values of the Jaccard coefficient, so that this time the x-interval extends to the value 1. These results confirm the strong positive correlation between values of the Jaccard coefficient between pairs' contact lists and social ties. An interesting difference is that this time, the threshold in the Jaccard coefficient above which we have a significant amount of social interaction is higher than in the previous case and there is no sharp transition. This can probably be explained by the fact that it seems to be easier to offer "friendship" on a social network than one's private phone number. Furthermore, on Facebook one can see (some of) the feeds of her friends' friends, while it is relatively easy for two perfect strangers to share one common acquaintance. A further lesson is that the right choice of the threshold is to some extent application specific, even though typical values seem to lie between 0.1 and 0.3 (see also results in Subsection 4.3.8.2).

4.3.8.1 Jaccard estimation performance

Reality Mining Project dataset. Figure 4.24 shows the behaviour of the Jaccard-Estimation Performance, as defined in the previous subsection, as a function of the threshold, both when 10 and 20 hash functions are used to estimate the Jaccard coefficient. In particular, the function has been computed in 5 points for 20 hash functions. Recall that each point represents the average taken over 10 independent runs of the algorithm.

More precisely, for $j = 1, \ldots, 5$, the runs of the j-th batch were executed with threshold value $0.05j$. For 10 hash functions we only report two points, since for values of the threshold above 0.1 the distance between the two curves becomes smaller and smaller. Results show that the algorithm that estimates the Jaccard coefficient using hash functions takes the same decisions as the one using the exact value of the Jaccard coefficient in the vast majority of cases. This means that, even under the stringent

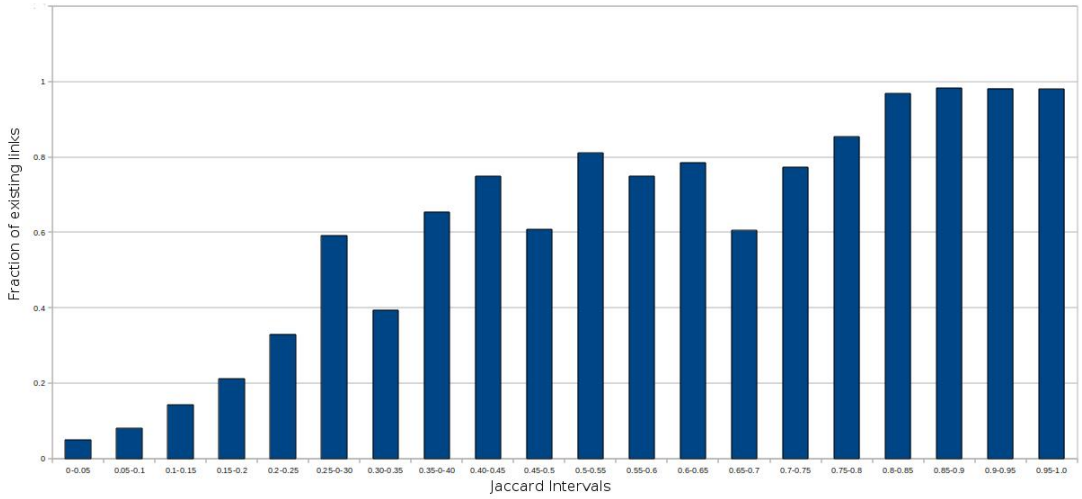


Figure 4.22: Existing linked pairs over total pairs, Jaccard Coefficient

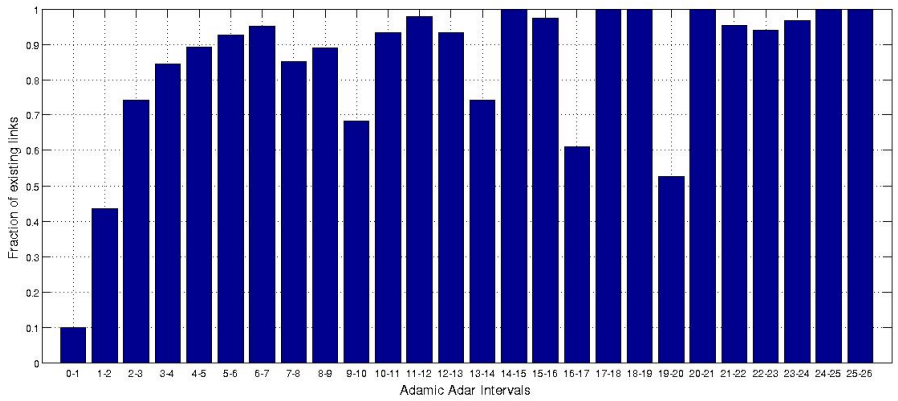


Figure 4.23: Existing linked pairs over total pairs, Adamic Adar coefficient

constraints for sketch sizes, we are able to follow the ideal algorithm pretty close, as far as the recommendation decision is concerned.

Facebook data set Figure 4.25 summarizes results for the Jaccard Estimation Performance index for the Facebook dataset. In this case, we considered a larger range for the number of hash functions, in order to also test the effect of increasing this number beyond values that are of practical interest in the SMS scenario but might be feasible in others. The general trend is clear: similarly to the previous case, for values of the threshold beyond 0.1, the hash-function based and the true Jaccard-based heuristic essentially provide the same recommendations. Two interesting features of these results are i) the slight decrease in JEP when the threshold grows from 0.8 to 0.9 and ii) the

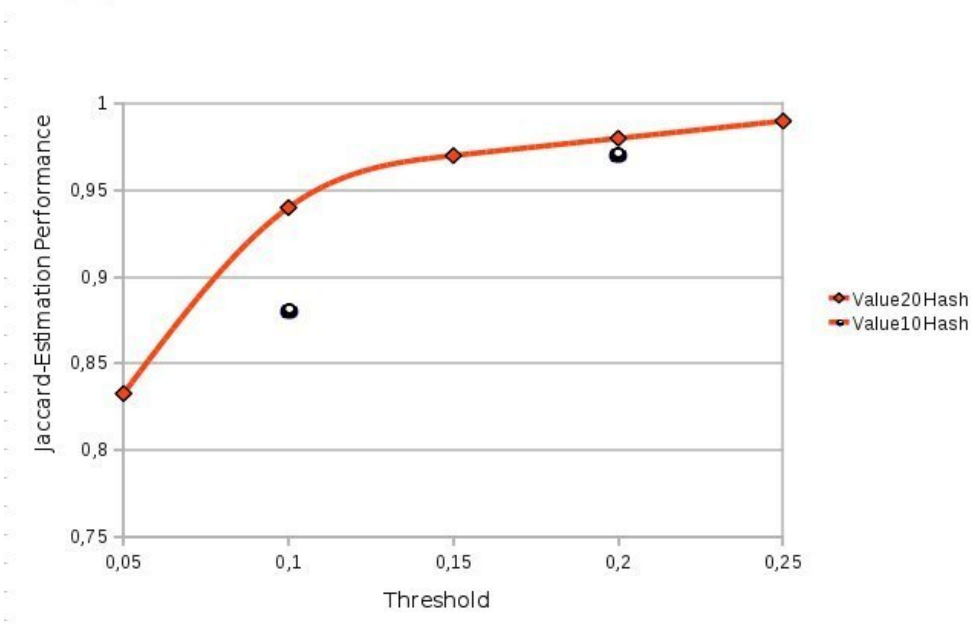


Figure 4.24: Jaccard-Estimation Performance for Reality Mining Project dataset.

decay in the JEP for the curve corresponding to 10 hash functions as the threshold grows from 0.1 to 0.2, which are discussed below.

Two opposite mechanisms are at work, that help explain the behaviour of the curves in Figure 4.25: on the one hand, as threshold grows, all heuristics are going to recommend new pairs when the interesection of the respective contact lists is higher in terms of Jaccard coefficient. On the other hand, as the latter grows, the hash function-based and the Jaccard-based heuristics are more likely to provide the same recommendations. At the same time, as threshold grows, both the hash-based and the Jaccard-based heuristics provide less recommendations overall. As a result, the marginal impact of cases where the two heuristics take different decisions increases. This latter effect explains the slight decay in the JEP curve when the threshold grows from 0.8 to 0.9. As for the decay in the JEP for the curve corresponding to 10 hash functions as the threshold grows from 0.1 to 0.2, the data show that the number of recommendations given by the Jaccard-based heuristic decreases much more than for the 10 hash function based heuristic in this interval, whereas the number of failures for the latter does not decrease as much. The reason for this fact is that for a threshold 0.2 the inaccuracy in the estimation of the Jaccard coefficient is still relatively high when using only 10 hash functions.

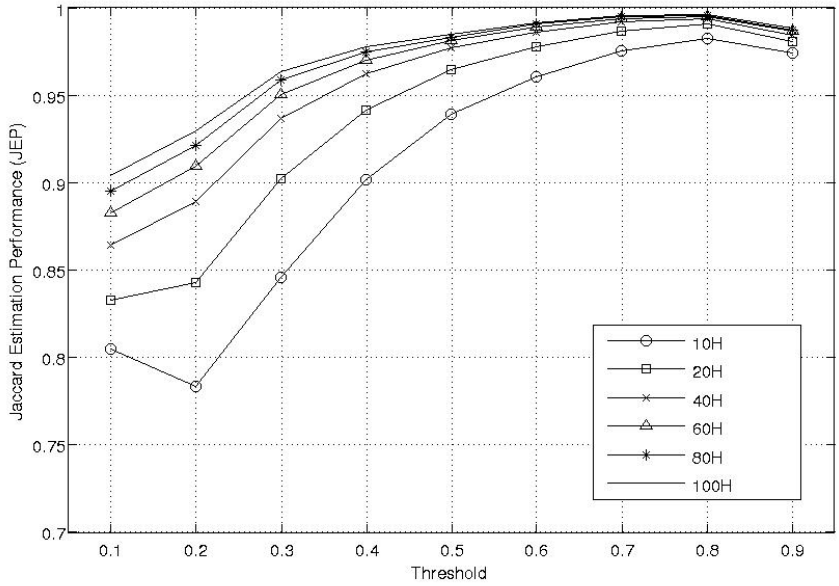


Figure 4.25: Jaccard Estimation Performance for Facebook dataset.

4.3.8.2 Quality of Recommendations

Figure 4.26 shows the effectiveness of our algorithms in predicting the existence of contacts in the social network of mobile users. In particular, Figure 4.26 is a scatter-plot showing the trade-off between precision and recall as the threshold and number of hash functions used vary[1]. For a better reading, values with $precision < 0.2$ or $recall < 0.2$ have been filtered out[2]. The following remarks are in order: i) the best trade-off between precision and recall is struck near the interval $[0.05, 0.1]$ of the threshold, both for the algorithm using exact Jaccard coefficient and for our heuristics; ii) For higher values precision increases and recall decreases, meaning that on one hand, a similarity beyond the threshold implies a contact with increasing probability, but we omit to recommend many contacts that fall below the threshold; iii) the values of precision/recall we obtain for the best choice of the threshold fall in the interval $[0.4, 0.6]$. Such values are indeed relatively high, since they refer to the prediction of really existing links; if we were only recommending links that already exist, there would be no point in providing recommendations.

In figure 4.27 we plotted a figure similar to the previous one, but this time we want to show that Jaccard coefficient has a behaviour that is comparable to that of Adamic Adar coefficient (AA in the following), one of the best performers. Please note that we used different values for selecting the thresholds By looking at the figure it is possible

[1]The threshold is represented as a label over each point in the scatterplot.

[2]This is the reason why only one point of the 10 hash algorithm is represented on the scatterplot.

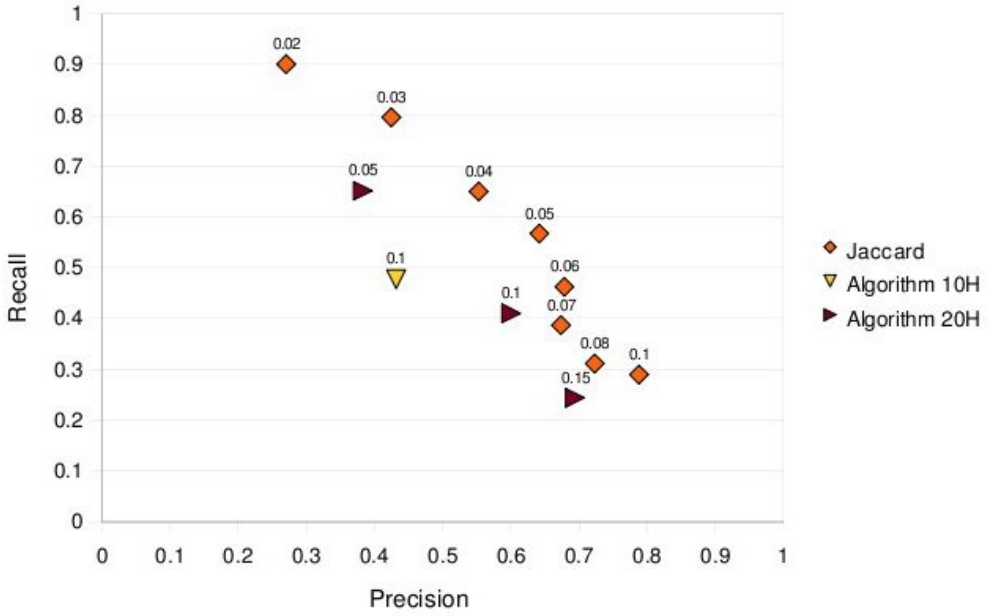


Figure 4.26: Precision vs recall.

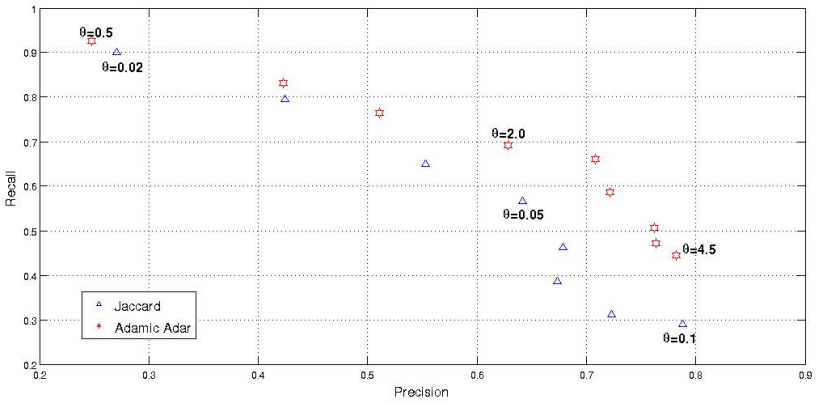


Figure 4.27: Adamic Adar vs Jaccard Precision and recall.

to note immediately that the general trend of Jaccard is very close to that of the AA coefficient. Even if the AA performs always slightly better than the Jaccard, the difference is so small that the Jaccard coefficient can be considered as a valid indicator of similarity between users, especially for the data-sets we considered.

Facebook dataset. Figure 4.29 shows that the trend emerging in the relatively small Reality Mining Project dataset is confirmed in the much larger Facebook one. Again, Figure 4.29 is a scatter-plot showing the trade-off between precision and recall as the

threshold and number of hash functions used vary. For the number of hash functions, we only report results for the cases $m = 10, 40, 100$ for the sake of readability. In the picture, θ is again the value of the threshold, with $\theta \in \{0.1, 0.2, \ldots, 0.9\}$ and increasing as we move from left to right. Then, for a given value of m or assuming the exact Jaccard coefficient is used to measure similarity, the behaviour is clear and intuitive as in the previous case: an increase in the threshold will improve precision (a link between two nodes is assumed when the overlap in their contact lists is higher) at the expense of recall (we are more likely to neglect links between pairs whose contact lists overlap below the threshold). In this perspective, in Figure 4.29, we emphasized three values of the threshold $\theta = 0.1, 0.3$ and 0.9 that are in some way representative of the different patterns arising in the results as the threshold varies. In particular, dashed ellipses group points that correspond to results for the same value of the threshold. If we do this for all values of the threshold, we note that points corresponding to the same value of the threshold tend to be grouped together, meaning that different heuristics perform similarly for the same value of threshold. In particular, this is always true when 20, 40 or 100 hash functions or the exact value of the Jaccard is used to measure similarity. Finally, we note that, even with 10 hash functions, setting the threshold to 0.3 allows to strike a very good balance between precision and recall for the Facebook dataset, with high values for both parameters and similar to those obtained for the Reality Mining Project dataset.

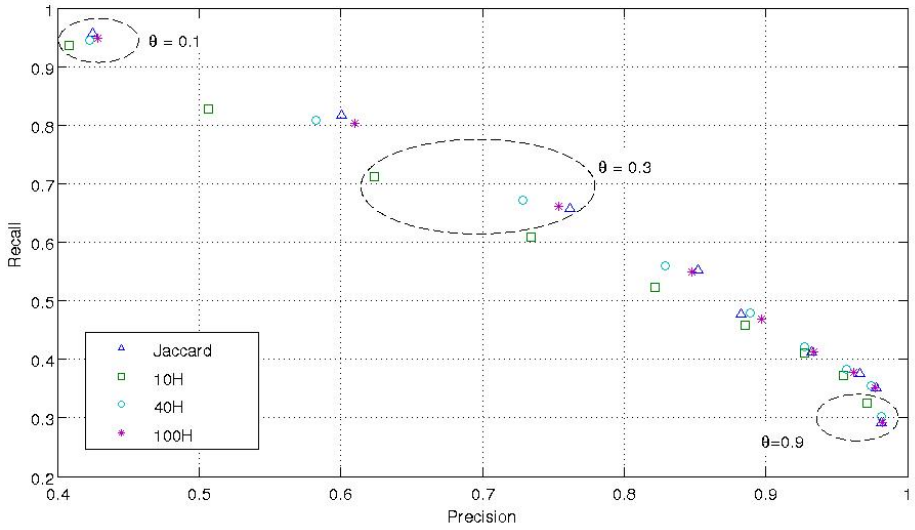


Figure 4.28: Precision and Recall for Facebook data set.

About our considerations on the validity of Jaccard coefficient, we reported also for this data-set the comparison between Jaccard coefficient and Adamic Adar. As we already noted for the previous data-set, both coefficients show a similar trend, with AA

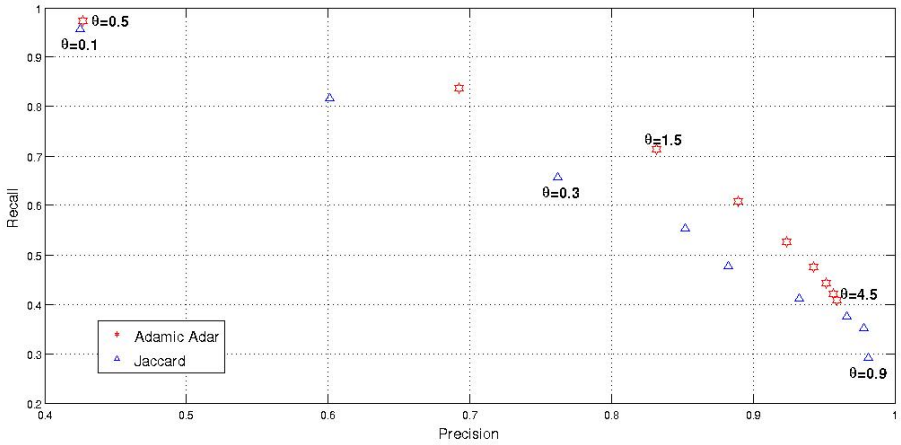


Figure 4.29: Adamic Adar vs Jaccard Precision and Recall for Facebook data set.

performing slightly better than Jaccard also in this case. But, again, we believe that the difference in terms of performance between the two indicators is so small that we can reasonably assume that Jaccard coefficient is a good estimator of user's similarity, and this confirms the validity of our approach.

4.3.9 Conclusions and future work

These results in our opinion provide an indication that our fully decentralized strategies might prove effective in providing recommendations of good quality. Furthermore, even though the Reality Mining Project dataset is relatively small and is likely to have some statistical bias, results indicate that, although in the same order of magnitude, the values of the threshold that correspond to the emergence of significant social interaction may in general vary depending on the application.

Moreover, our results indicate that similarity of users' contact lists already provides a powerful feature to provide useful contat recommendations. An obvious direction for future research is to extend our techniques to provide sketches of semantically reacher user profiles, possibly including heterogeneous features.

4.4 Mobile Dynamic Social Networks and Internet Art

An interesting and unexpected corollary application we developed during the MACRO collection of mobility traces was the artwork (143). We already said previously that one of the social contexts were we collected real traces was the opening of an artwork exhibition at the MACRO museum (Museum of Contemporary Art of Rome). Thanks to the collaboration with contemporary artist Miltos Manetas, we realized the artwork "300visitors", that is currently available at the cited web address. In this thesis we believe it is important to spend a few words on the description of the technological details of the application, since it is not easy as it may seem at a first glance. Different technologies were combined together to make the final visualization available, as we will see in the following description.

4.4.1 Data collection infrastructure

The data collection infrastructure was described in the previous chapter, when discussing about traces; it is sufficient to recall here that the data sent by tags worn by visitors were collected thanks to a few wireless bridges we realized to reach the collection server.

4.4.2 Data visualization

As we described in section 3.3, the data transmitted to the server are written on a text file according to a defined format. For the realization of 300visitors artwork, we needed an additional feature for our system: the data to be projected on the huge screen should be almost "real-time", as our intention was to show the movements and interactions of the crowd as they were taking place. For this reason, we developed a Java parser that was able to automatically detect new lines in the text log file, and to create new json objects with relevant informations (readers addressess, list of seen tags and so on) for each different timestamp. At this point the visualization was totally managed by an html and jsp page, with the help of a class written in "Processing" (144) programming language. Processing is an open source programming language and environment that ease the development of graphical applications. Thanks to many libraries developed by volunteer contributors, it is able to support a variety of applications; it was the perfect choice for our application. The main idea of the artwork is that of visualizing interactions and persons with a digital icon. We decided to use a set of images of desktop pointers, chosen at random for each user (see figure 4.30).

We then used the positions of the readers as "anchors", so that a user is placed in a random position around the reader that sent the message. When a contact message is detected, the pointers representing the users become bigger and stop moving, and a short music piece starts. An image of a sample interaction is depicted in figure 4.31.

When the number of social contacts grows, many different musics start playing together, creating kind of a symphony, generated in real time by the movement and

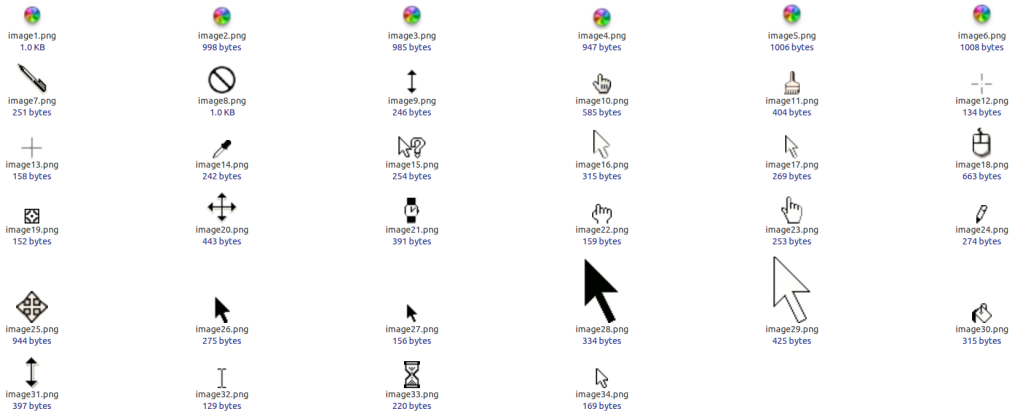


Figure 4.30: Images of pointers

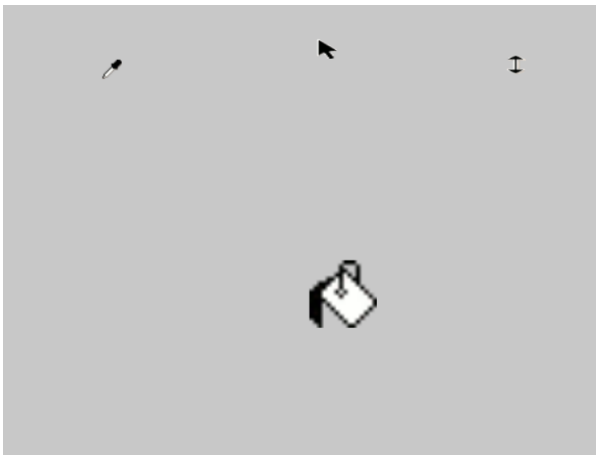


Figure 4.31: A sample interaction with a pointer becoming bigger

contacts between persons. A screenshot of the application when $\approx$ 100 persons are moving around is depicted in figure 4.32

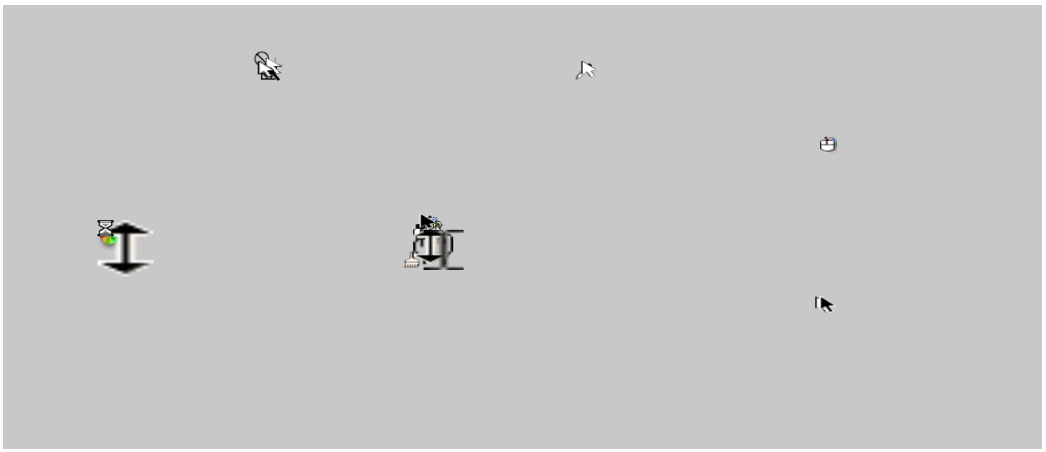


Figure 4.32: Sample interactions with a high number of users

The artwork we created was appreciated by all the participants, and the installation will be presented in a forthcoming conference (Artsit 2013, (145)) focused on the relationship between arts and information technology for which our artwork was selected.

5

Conclusions

Social networking platforms are quickly becoming fundamental ingredients of our everyday lives, a trend that also resulted in a growing interest toward social networks from computer scientists. A deep analysis of social interactions and the ability of predicting social behaviors could be of fundamental importance in different areas, ranging from viral advertising to recommendations and so on. Many previous works focused their attention on the analysis of online social platforms, like Facebook, Google+, LinkedIn and similar services, mainly because so far the availability of collections of real traces from real social populations was quite limited. In this thesis, we focused on dynamic social networks originating from the interaction overtime of mobile wireless agents in the physical space. Thanks to technological advancements, several datasets began to be available for public online access, but as we extensively discussed in chapter 3, some of those data were not appropriately representing a dynamic social network. We underlined how it is possible to look at a population of individuals like a dynamic social network, where connections between different nodes of the network may vary over time according to the movement of the nodes themselves (which represent real persons). We reviewed the main distinguishing features of dynamic social networks with respect to static networks, and we presented a few examples of real dynamic wireless networks. We identified in "Opportunistic" and "Efficiency" the 2 fundamental keywords describing the communication we must provide in the dynamic mobile social networks we considered. The communication is in fact assumed to be "Opportunistic" because we cannot assume a stable path connecting any 2 nodes of the networks; links may vary over time according to the movement of the nodes themselves. The communication must also be efficient, since we are considering tiny devices with limited computational capabilities. To this purpose, in chapter 2 we reviewed some of the existing theoretical models for generating dynamic networks, and some mobility models to regulate the movement of elements of the network. We explained why some models could fit our needs, and why some others are not adequate. We also remarked how the use of synthetic models for generating interactions between elements of the populations can help in testing several

different scenarios in a fast and easy way, at the cost of a certain distance from the real behavior of a population. The use of real movement traces, on the other hand, requires a long work to set up an appropriate infrastructure for trace recording, but the algorithm could then be tested in an extremely realistic situation. For this reason, we reviewed in chapter 3 the main adopted technologies for recording movement traces, and we presented the one we decided to adopt ourselves for our trace recording. We presented the whole infrastructure we set up, and we showed how OpenBeacon tags currently represent the best available solution for recording fine-grained face-to-face interactions between persons. In chapter 3 we also presented a short excursus in which we analyzed some of the available simulation platforms. Simulations, in fact, are not only composed of movement, but also of several other element (like radio propagation, simulation of the device itself and so on) that must be properly evaluated to understand how reliable is the result produced by a simulator. In chapter 4 we finally illustrated how it is possible to reproduce a full theoretical computational model (population protocols) in a real network of tiny devices. We showed that the implementation is quite accurate, and we paved the way for the implementation of other theoretical computational models. We also showed how, by using devices slightly powerful than the OpenBeacon tags, it is possible to solve an interesting and diffused problem like distributely counting the distinct elements of a dynamic network. An additional step was showing the possibility of accurately predicting links in a social network of sms-users by using mobile phones and sketching algorithms. A last, unexpected, evolution of our research activity was the collaboration with a contemporary internet artist, which took to the creation of an artwork representing a visual representation of the mobile dynamic social network formed during our recording of traces. All the previous results, thus, showed that it is perfectly possible to take to real contexts some problems that are usually considered only from a theoretical point of view.

Future Works For what concerns the work we are planning for the future, we are currently identifying other interesting problems we would like to solve using tiny devices, and we are also trying to optimize the distinct counting problem so that it can be solved also on very low power devices like the openbeacons tags (recall that actually we needed at least Telosb nodes to solve it). We are also exploring new research directions to ease the collection of traces of social interactions, for example by developing an appropriate application for smartphones, so that users are not forced to wear additional devices, and so that interaction traces could be collected with a very simple infrastructure. Another research direction started from the use of OpenBeacon tags is the development of an indoor navigation system. We already realized a preliminar test in our university, where volunteer participants were localized in different points of interest inside the department, and upon request they had the possibility of receiving indications on how to reach a specific place. This application was tested and working fine, and our work is actually on the use of smartphones to localize users, to free them

from the necessity of wearing an OpenBeacon tag. The collaboration with the contemporary artist is also continuing, and we are thinking on how to improve the localization of persons indoor, to create an artwork were each participant leaves a "Trace" of its passage on the screen.

References

[1] Kevin Lewis, Jason Kaufman, Marco Gonzalez, Andreas Wimmer, and Nicholas Christakis. **Tastes, ties, and time: A new social network dataset using Facebook.com**, 2009. 1

[2] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. **Measurement and analysis of online social networks**. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM. 1

[3] **http://www.sociopatterns.org/**. 1, 35

[4] **Reality Mining project**. http://reality.media.mit.edu/. 1, 33, 107

[5] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. **Pocket Switched Networks and Human Mobility in Conference Environments**. In *Proceedings of ACM SIGCOMM first workshop on delay tolerant networking and related topics*, 2005. 1, 21, 31

[6] Guoqiang Mao and B.D.O. Anderson. **Graph Theoretic Models and Tools for the Analysis of Dynamic Wireless Multihop Networks**. pages 1 –6, apr. 2009. 1

[7] Cemal Cagatay Bilgin and Blent Yener. **Dynamic Network Evolution: Models, Clustering, Anomaly Detection**. 5, 6

[8] **Internet Movie Database**. 5

[9] **Computer Science Bibliography**. 5

[10] D.J.Watts and S.H.Strogatz. **Collective dynamics of small-world networks**. In *Nature*, 1998. 6, 11

[11] F. Harary and G. Gupta. **Dynamic graph models**. *Math. Comput. Model.*, **25**(7):79–87, April 1997. 7

[12] Chen Avin, Michal Koucký, and Zvi Lotker. **How to Explore a Fast-Changing World (Cover Time of a Simple Random Walk on Evolving Graphs)**. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, ICALP '08, pages 121–132, Berlin, Heidelberg, 2008. Springer-Verlag. 7

[13] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. **Distributed computation in dynamic networks**. In *STOC '10: Proceedings of the 42nd ACM symposium on Theory of computing*, pages 513–522, New York, NY, USA, 2010. ACM. 7

[14] P. Erdös and A Rényi. **On the Evolution of Random Graphs**. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960. 9

[15] Andrea E.F. Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. **Flooding time in edge-Markovian dynamic graphs**. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, PODC '08, pages 213–222, New York, NY, USA, 2008. ACM. 10

[16] Leonard J. Schulman, editor. *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. ACM, 2010. 10, 68

[17] Reka Albert Albert Barabasi. **Dynamic Network Evolution: Models, Clustering, Anomaly Detection**. 12

[18] Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. **Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication**. In *In PKDD*, pages 133–145. Springer, 2005. 14

[19] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. **Graph evolution: Densification and shrinking diameters**. *ACM Trans. Knowl. Discov. Data*, **1**(1), March 2007. 14

[20] Jihwang Yeo, David Kotz, and Tristan Henderson. **CRAWDAD: a community resource for archiving wireless data at Dartmouth**. *SIGCOMM Comput. Commun. Rev.*, **36**(2):21–22, April 2006. 15

[21] Mirco Musolesi and Cecilia Mascolo. **Mobility Models for Systems Evaluation. A Survey**. In Benoit Garbinato, Hugo Miranda, and Luis Rodrigues, editors, *Middleware for Network Eccentric and Mobile Applications*, pages 43–62. Springer, February 2009. 16

[22] Josh Broch, David A. Maltz, David B. Johnson, Yih chun Hu, and Jorjeta Jetcheva. **A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols**. pages 85–97, 1998. 16

[23] Fan Bai and Ahmed Helmy. **Chapter 1 A SURVEY OF MOBILITY MODELS in Wireless Adhoc Networks**. 17

[24] Christian Bettstetter. **Mobility modeling in wireless networks: categorization, smooth movement, and border effects**. *SIGMOBILE Mob. Comput. Commun. Rev.*, **5**(3):55–66, July 2001. 17

[25] Douglas M. Blough, Giovanni Resta, and Paolo Santi. **A statistical analysis of the long-run node spatial distribution in mobile ad hoc networks**. In *Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, MSWiM '02, pages 30–37, New York, NY, USA, 2002. ACM. 17

[26] Elizabeth M. Royer, P. Michael Melliar smith Y, and Louise E. Moser Y. **An analysis of the optimum node density for ad hoc mobile networks**. In *In Proceedings of the IEEE International Conference on Communications*, pages 857–861, 2001. 18

[27] Christian Bettstetter. **Smooth is better than sharp: a random mobility model for simulation of wireless networks**. In *Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, MSWIM '01, pages 19–27, New York, NY, USA, 2001. ACM. 18

[28] B. Liang and Z.J. Haas. **Predictive distance-based mobility management for PCS networks**. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, **3**, pages 1377–1384 vol.3, Mar. 18

[29] Tracy Camp, Jeff Boleng, and Vanessa Davies. **A survey of mobility models for ad hoc network research**. *Wireless Communications and Mobile Computing*, **2**(5):483–502, 2002. 19

[30] Jing Tian, Jörg Hähner, Christian Becker, Illya Stepanov, and Kurt Rothermel. **Graph-Based Mobility Model for Mobile Ad Hoc Network Simulation**. In *Annual Simulation Symposium*, pages 337–344, 2002. 20

[31] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. **Scenario-based performance analysis of routing protocols for mobile ad-hoc networks**. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 195–206, New York, NY, USA, 1999. ACM. 20

[32] Amit Jardosh, Elizabeth M. Belding-Royer, Kevin C. Almeroth, and Subhash Suri. **Towards realistic mobility models for mobile ad hoc networks**. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, MobiCom '03, pages 217–229, New York, NY, USA, 2003. ACM. 20

[33] Sokol Kosta, Alessandro Mei, and Julinda Stefa. **Small World in Motion (SWIM): Modeling Communities in Ad-Hoc Mobile Networking**. In *SECON*, pages 10–18, 2010. 20, 21

[34] Mirco Musolesi and Cecilia Mascolo. **A community based mobility model for ad hoc network research**. In *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, REALMAN '06, pages 31–38, New York, NY, USA, 2006. ACM. 21, 22

[35] Mirco Musolesi, Stephen Hailes, and Cecilia Mascolo. **An ad hoc mobility model founded on social network theory**. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '04, pages 20–24, New York, NY, USA, 2004. ACM. 21

[36] M. E. J. Newman and M. Girvan. **Finding and evaluating community structure in networks**. *Physical Review*, **E 69**(026113), 2004. 23

[37] Xiaoyan Hong, Mario Gerla, Guangyu Pei, and Ching-Chuan Chiang. **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*, MSWiM '99, pages 53–60, New York, NY, USA, 1999. ACM. 24

[38] Miguel Sánchez López and Pietro Manzoni. **ANEJOS: a Java based simulator for ad hoc networks**. *Future Generation Comp. Syst.*, **17**(5):573–583, 2001. 24, 25

[39] Cristian Tuduce and Thomas R. Gross. **A mobility model based on WLAN traces and its validation**. In *INFOCOM*, pages 664–674, 2005. 26

[40] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. **Pocket switched networks: Real-world mobility and its consequences for opportunistic forwarding**. Technical report, 2005. 30, 31

[41] Tristan Henderson, David Kotz, and Ilya Abyzov. **The changing usage of a mature campus-wide wireless network**. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, MobiCom '04, pages 187–201, New York, NY, USA, 2004. ACM. 30

[42] David Kotz and Kobby Essien. **Analysis of a campus-wide wireless network**. *Wirel. Netw.*, **11**(1-2):115–133, January 2005. 30

[43] Marvin McNett and Geoffrey M. Voelker. **Access and mobility of wireless PDA users**. *SIGMOBILE Mob. Comput. Commun. Rev.*, **9**(2):40–55, April 2005. 30

[44] Nathan Eagle and Alex Pentland. **Reality mining: sensing complex social systems**. *Personal and Ubiquitous Computing*, **10**(4):255–268, 2006. 33, 107

[45] Kay Römer and Friedemann Mattern. **The Design Space of Wireless Sensor Networks**. *IEEE Wireless Communications*, **11**(6):54–61, December 2004. 33

[46] **http://www.openbeacon.org/**. 35

[47] Ciro Cattuto, Wouter Van den Broeck, Alain Barrat, Vittoria Colizza, Jean-Franois Pinton, and Alessandro Vespignani. **Dynamics of Person-to-Person Interactions from Distributed RFID Sensor Networks**. *PLoS ONE*, **5**(7):e11596, 07 2010. 37

[48] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. **Gephi: An Open Source Software for Exploring and Manipulating Networks**, 2009. 41

[49] **http://gephi.org/**. 41, 45

[50] **http://gexf.net/format/primer.html**. 41

[51] **http://www.json.org/**. 46

[52] Ugo Maria Colesanti, Carlo Crociani, and Andrea Vitaletti. **On the accuracy of OMNeT++ in the wireless sensornet works domain: simulation vs. testbed**. In *PE-WASUN '07: Proceedings of the 4th ACM workshop on Performance evaluation of wireless ad hoc, sensor,and ubiquitous networks*, pages 25–31, New York, NY, USA, 2007. ACM. 50

[53] Svilen Ivanov, Andr Herms, and Georg Lukas. **Experimental validation of the ns-2 wireless model using simulation, emulation, and real network**. In *In 4th Workshop on Mobile Ad-Hoc Networks (WMAN07*, pages 433–444, 2007. 50

[54] David Kotz, Calvin Newport, Robert S. Gray, Jason Liu, Yougu Yuan, and Chip Elliott. **Experimental evaluation of wireless simulation assumptions**. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '04, pages 78–82, New York, NY, USA, 2004. ACM. 50

[55] Jisun Lee, Matt Perkins, Spyros Kyperountas, and Youngmin Ji. **RF Propagation Simulation in Sensor Networks**. In *Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications*, SENSORCOMM '08, pages 604–609, Washington, DC, USA, 2008. IEEE Computer Society. 50

[56] Nithya Ramanathan, Eddie Kohler, and Deborah Estrin. **Towards a debugging system for sensor networks**. *Int. J. Netw. Manag.*, **15**(4):223–234, July 2005. 51

[57] **OMNeT++**. http://www.omnetpp.org. 51

[58] **WISEBED**. http://www.wisebed.eu/. 51

[59] **CitySense**. http://www.citysense.net/. 51

[60] **WUSTL**. www.cse.wustl.edu/wsn/index.php?title=Testbed. 51

[61] P.Swieskowski G.W.Allen, M.Welsh. **Motelab: a Wireless Sensor Network Testbed**. In *IPSN'05*, 2005. http://motelab.eecs.harvard.edu/. 51, 55, 94

[62] **Kansei**. http://ceti.cse.ohio-state.edu/kansei/. 51

[63] E. Ould-Ahmed-Vall, G.F. Riley, B.S. Heck, and D. Reddy. **Simulation of large-scale sensor networks using GTSNetS**. In *MASCOTS '05*, pages 211–218, Sept. 2005. 51

[64] Alexander Kröller, Dennis Pfisterer, Carsten Buschmann, Sándor P. Fekete, and Stefan Fischer. **Shawn: A new approach to simulating wireless sensor networks**. *CoRR*, **abs/cs/0502003**, 2005. 51, 53

[65] Sergey Fomel and Jon F. Claerbout. **Guest Editors' Introduction: Reproducible Research**. *Computing in Science and Engineering*, **11**(1):5–7, 2009. 51

[66] David Cavin, Yoav Sasson, and André Schiper. **On the accuracy of MANET simulators**. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, POMC '02, pages 38–43, New York, NY, USA, 2002. ACM. 51

[67] E. Weingartner, H. vom Lehn, and K. Wehrle. **A Performance Comparison of Recent Network Simulators**. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–5, June 2009. 52

[68] Luc Hogie, Pascal Bouvry, and Frédéric Guinand. **An Overview of MANETs Simulation**. *Electron. Notes Theor. Comput. Sci.*, **150**(1):81–101, March 2006. 52

[69] Ns-2 Simulator. ***www.isi.edu/nsnam/ns/***. 52, 55

[70] Dimosthenis Pediaditakis Hai N. Pham and Athanassios Boulis. **From simulations to real deployment in WSNs and back**. *t2pWSN'2007*, 2007. 53

[71] **sensLAB**. http://www.senslab.info/index.php/Testbed. 53

[72] **https://www.itm.uni-luebeck.de/ShawnWiki/index.php/Shawn_Introduction**. 53, 92

[73] **http://ccl.northwestern.edu/netlogo/**. 54, 73

[74] Lorenzo Bergamini, Carlo Crociani, Andrea Vitaletti, and Michele Nati. **Validation of WSN simulators through a comparison with a real testbed**. In *Proceedings of the 7th ACM workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, PE-WASUN '10, pages 103–104, New York, NY, USA, 2010. ACM. 55

[75] Hai N. Pham, Dimosthenis Pediaditakis, and Athanassios Boulis. **From Simulation to Real Deployments in WSN and Back.** In *WOWMOM*, pages 1–6. IEEE, 2007. 55

[76] Theodore S Rappaport. *Wireless Communications: Principles and Practice.* Prentice Hall, 1996. 55

[77] **CC2420 Datasheet**. http://inst.eecs.berkeley.edu/ cs150/Documents/CC2420.pdf. 57

[78] **TmoteSky DataSheet**. Moteiv. 57, 62

[79] www.tinyos.net. 57, 94

[80] Rangwala et al. Xu. **A wireless sensor network for structural monitoring**. In *Proceedings of SenSys*, 2004. 59

[81] Hoang Maimour, Pham. **A congestion control framework for handling video-surveillance traffic on WSN**. In *Proceedings of International Conference on Computational Science and Engineering*, pages 943–948, 2009. 59

[82] Bischoff et al. Meyer. **Wireless Sensor Networks for long term structural health monitoring**. 2006. 59

[83] J. Lee, M. Perkins, S. Kyperountas, and Youngmin Ji. **RF Propagation Simulation in Sensor Networks**. In *SENSORCOMM '08*. 59, 60

[84] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. **The computational power of population protocols**. *Distributed Computing*, **20**(4):279–304, November 2007. 68, 70

[85] James Aspnes and Eric Ruppert. **An introduction to population protocols**. *Bulletin of the European Association for Theoretical Computer Science*, **93**:98–117, October 2007. 69

[86] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. **Computation in networks of passively mobile finite-state sensors**. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 290–299, New York, NY, USA, 2004. ACM. 69, 70, 75

[87] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. **Computation in networks of passively mobile finite-state sensors**. *Distributed Computing*, pages 235–253, 2006. 69

[88] James Aspnes and Eric Ruppert. **An introduction to population protocols**. In *Middleware for Network Eccentric and Mobile Applications, Benoît Garbinato, Hugo Miranda, and Luís Rodrigues, eds.*, pages 97–120. Springer-Verlag, 2009. 70, 75

[89] Dana Angluin, James Aspnes, and David Eisenstat. **Stably computable predicates are semilinear**. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 292–299, New York, NY, USA, 2006. ACM Press. 70

[90] Spanier E.H. Ginsburg S. **Presburger formulas and languages.** 70

[91] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. **The computational power of population protocols**. *Distributed Computing*, **20**(4):279–304, 2007. 72

[92] Luca Becchetti, Lorenzo Bergamini, Francesco Ficarola, and Andrea Vitaletti. **Population protocols on real social networks**. In *PE-WASUN*, pages 17–24, 2012. 73

[93] Luca Becchetti, Lorenzo Bergamini, Francesco Ficarola, and Andrea Vitaletti. **Population protocols on real social networks**. In *SNS*, page 15, 2012. 73

[94] Luca Becchetti, Lorenzo Bergamini, Francesco Ficarola, Francesco Salvatore, and Andrea Vitaletti. **First Experiences with the Implementation and Evaluation of Population Protocols on Physical Devices**. In *GreenCom*, pages 335–342, 2012. 79

[95] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. **When birds die: making population protocols fault-tolerant**. In *Proceedings of the Second IEEE international conference on Distributed Computing in Sensor Systems*, DCOSS'06, pages 51–66, Berlin, Heidelberg, 2006. Springer-Verlag. 81

[96] Kyu young Whang, Brad T. Vander-zanden, and Howard M. Taylor. **A linear-time probabilistic counting algorithm for database applications**. *ACM Transactions on Database Systems*, **15**:208–229, 1990. 85

[97] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. **Selectivity and Cost Estimation for Joins Based on Random Sampling**. *J. Comput. Syst. Sci.*, **52**(3):550–569, 1996. 85

[98] Philippe Flajolet and G. Nigel Martin. **Probabilistic Counting Algorithms for Data Base Applications**. *J. Comput. Syst. Sci.*, **31**(2):182–209, 1985. 86, 87, 88, 90, 91, 92

[99] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. **Counting Distinct Elements in a Data Stream**. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, RANDOM '02, pages 1–10, London, UK, UK, 2002. Springer-Verlag. 86

[100] David E. Culler, Deborah Estrin, and Mani B. Srivastava. **Guest Editors' Introduction: Overview of Sensor Networks**. *IEEE Computer*, **37**(8):41–49, 2004. 86

[101] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. **TAG: a Tiny AGgregation service for ad-hoc sensor networks**. *SIGOPS Oper. Syst. Rev.*, **36**(SI):131–146, December 2002. 86

[102] S. Muthukrishnan. **Data Streams: Algorithms and Applications**. *Foundations and Trends in Theoretical Computer Science*, **1**(2), 2005. 87, 90

[103] A. Akella, A. Bharambe, M. Reiter, and S. Seshan. **Detecting DDoS Attacks on ISP Networks**. In *Proceedings of ACM SIGMOD/PODS Workshop on Management and Processing of Data Streams (MPDS) FCRC 2003*, San Diego, CA, June 2003. 87

[104] Cristian Estan, George Varghese, and Michael E. Fisk. **Bitmap algorithms for counting active flows on high-speed links**. *IEEE/ACM Trans. Netw.*, **14**(5):925–937, 2006. 87

[105] Jeffrey Considine, Feifei Li, George Kollios, and John W. Byers. **Approximate Aggregation Techniques for Sensor Databases**. In *ICDE*, pages 449–460, 2004. 87

[106] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. **Synopsis diffusion for robust aggregation in sensor networks**. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 250–262, New York, NY, USA, 2004. ACM. 87

[107] Jeffrey Considine, Marios Hadjieleftheriou, Feifei Li, John W. Byers, and George Kollios. **Robust approximate aggregation in sensor data management systems**. *ACM Trans. Database Syst.*, **34**(1), 2009. 87

[108] Piotr Indyk and David P. Woodruff. **Tight Lower Bounds for the Distinct Elements Problem**. In *FOCS*, pages 283–288, 2003. 87

[109] Michael Resvanis and Ioannis Chatzigiannakis. **Experimental Evaluation of Duplicate Insensitive Counting Algorithms**. In *Panhellenic Conference on Informatics*, pages 60–64, 2009. 88, 89

[110] Marianne Durand and Philippe Flajolet. **Loglog Counting of Large Cardinalities**. In *In ESA*, pages 605–617, 2003. 89

[111] Graham Cormode, Srikanta Tirthapura, and Bojian Xu. **Time-decaying sketches for sensor data aggregation**. In *PODC*, pages 215–224, 2007. 89

[112] Lorenzo Bergamini, Luca Becchetti, and Andrea Vitaletti. **Privacy-Preserving Environment Monitoring in Networks of Mobile Devices**. In *Networking Workshops*, pages 179–191, 2011. 90

[113] Li Considine, Kollios. **Approximate aggregation techniques for sensor Databases**. In *Proceedings of ICDE*, 2004. 90, 92

[114] David B. Johnson and David A. Maltz. *Mobile Computing - Dynamic source routing in ad hoc wireless networks*, chapter 5, pages 153 – 181. Kluwer Academic Publishers, 1996. 91

[115] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary Anderson. **Synopsis diffusion for robust aggregation in sensor networks**. *ACM Trans. Sen. Netw.*, **4**(2):1–40, 2008. 91

[116] Jeffrey Considine, Feifei Li, George Kollios, and John W. Byers. **Approximate Aggregation Techniques for Sensor Databases**. In *ICDE*, pages 449–460, 2004. 91

[117] Marios Hadjieleftheriou, John Byers, and George Kollios. **Robust Sketching and Aggregation of Distributed Data Streams**. Technical Report 2005-011, CS Department, Boston University, 2005. 91

[118] Graham Cormode, S. Muthukrishnan, and Wei Zhuang. **What's Different: Distributed, Continuous Monitoring of Duplicate-Resilient Aggregates on Data Streams**. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006*. IEEE Computer Society, 2006. 92

[119] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. **Counting Distinct Elements in a Data Stream**. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, pages 1–10, Cambridge, Ma, USA, 2002. Springer-Verlag. 92

[120] **Tmote sky datasheet**. http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf. 94

[121] L.Bergamini-U.Colesanti L.Filipponi A.Vitaletti G.Persiano E.Baglioni, L.Becchetti. **A lightweight privacy preserving SMS-based recommendation system for mobile users**. In *Proceedings of RecSys 2010*, September 26-30, Barcelona 2010. 97

[122] Becchetti L., Bergamini L., Colesanti U., Filipponi L., Vitaletti A., and Persiano G. **A lightweight privacy preserving SMS-based recommendation system for mobile users**. To appear in Knowledge and Information Systems, Springer, 2013. 97

[123] **Market intelligence on messaging and marketing**. http://www.lloydsmiu.com/pdf/Jun-2005/16/mma_sample_issue_june.pdf, 2009. 97

[124] **Jyngle Web site. http://www.jyngle.com/**, 2008. 97

[125] **Peekamo Web site: http://peekamo.com/**, 2008. 97

[126] Gartner Consulting. **Gartner Says Mobile Messages to Surpass 2 Trillion Messages in Major Markets in 2008**, 2008. 97

[127] William Aiello, Fan Chung, and Linyuan Lu. **Random Evolution in Massive Graphs**. In Bob Werner, editor, *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 510–521, 2001. 98

[128] William Aiello, Fan Chung, and Linyuan Lu. **A random graph model for massive graphs**. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180. ACM, 2000. 98

[129] M. E. J. Newman. **The structure and function of complex networks**, March 2003. 98, 100, 108

[130] David Liben-Nowell and Jon Kleinberg. **The link-prediction problem for social networks**. *J. Am. Soc. Inf. Sci. Technol.*, **58**(7):1019–1031, 2007. 98, 100

[131] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. **Min-wise independent permutations**. In *ACM Symposium on the Theory of Computing*, New York, NY, USA, 1998. 100, 103

[132] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. **Syntactic clustering of the Web**. In *Proc. of the World Wide Web Conference*, 1997. 100, 103

[133] Piotr Indyk. **A Small Approximately Min-Wise Independent Family of Hash Functions**. In *SODA*, 1999. 100

[134] Tom Bohman, Colin Cooper, and Alan M. Frieze. **Min-Wise Independent Linear Permutations**. *Electr. J. Comb*, **7**, 2000. 100

[135] Robin Dunbar. *Grooming, Gossip, and the Evolution of Language*. Harvard University Press, 1998. 103

[136] Petros Zerfos, Xiaoqiao Meng, Starsky H.Y Wong, Vidyut Samanta, and Songwu Lu. **A study of the short message service of a nationwide cellular network**. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 263–268, New York, NY, USA, 2006. ACM. 103

[137] Pascal Paillier. **Public-Key Cryptosystems Based on Composite Degree Residuosity Classes**. In Jacques Stern, editor, *Advances in Cryptology EUROCRYPT 99*, **1592** of *Lecture Notes in Computer Science*, pages 223–238. Springer Berlin / Heidelberg, 1999. 104

[138] Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. **Practical Recommendations on Crawling Online Social Networks**. *IEEE Journal on Selected Areas in Communications*, **29**(9):1872–1892, 2001. 107

[139] Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. **On unbiased sampling for unstructured peer-to-peer networks**. *IEEE/ACM Trans. Netw.*, **17**:377–390, April 2009. 107

[140] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger. **Sampling Techniques for Large, Dynamic Graphs**. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1 –6, april 2006. 107

[141] Lada A Adamic and Eytan Adar. % bf Friends and neighbors on the Web. *Social Networks*, **25**(3):211 – 230, 2003. 110

[142] David Liben-Nowell and Jon Kleinberg. **The link prediction problem for social networks**. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 556–559, New York, NY, USA, 2003. ACM. 110

[143] F.Ficarola M.Manetas A.Vitaletti L.Becchetti, L.Bergamini. 119

[144] **Processing programming language**. 119

[145] Becchetti L., Bergamini L., Ficarola F., Vitaletti A., and Manetas M. **300 Visitors**. To appear in International Journal of Arts and Technology (Inderscience), 2013. 121