# Data Backup for Mobile Nodes : a Cooperative Middleware and an Experimentation Platform [*]

Marc-Olivier Killijian and Matthieu Roy

CNRS ; LAAS ; 7 avenue du colonel Roche; F-31077 Toulouse, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France
Contact: `name.surname@laas.fr`
http://theresumeexperience.blogspot.com/

**Abstract.** In this paper, we present a middleware for dependable mobile systems and an experimentation platform for its evaluation. Our middleware is based on three original building blocks: a Proximity Map, a Trust and Cooperation Oracle, and a Cooperative Data Backup service. A Distributed Black-box application is used as an illustrative application of our architecture, and is evaluated on top of our mobile experimental platform.

## 1 Problem Statement

Finding the proper abstractions to design middleware for the provision of dependable distributed applications on mobile devices remains a big challenge [1]. The number of mobile communicating devices one can meet in every-day life is dramatically increasing: mobile phones, PDAs, handheld GPS, laptops and notebooks, portable music and video players. Those devices benefit from an amazing number of sensors and communication interfaces. The interconnection of these systems does not only result in a huge distributed system. New technical and scientific challenges emerge due to the mobility of users and of their devices, or due to the massive scale of uncontrolled devices that constantly connect and disconnect, fail, etc. To handle those systems' dynamics, cooperation-based approaches *à la* peer-to-peer seem attractive. An important question is thus to know if and how can we design a sound middleware that offers useful building blocks for this type of system. Another crucial question is to study how we can correctly evaluate those highly mobile and dynamic systems.

This paper addresses these two questions: we present a *middleware architecture* dedicated to the provision of cooperative data backup on mobile nodes

[*] This work was partially supported by the Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) under the ARUM internal project, French National Center for Scientific Research (CNRS), by the MoSAIC project (ACI S&I, French national program for Security and Informatics), the European Hidenets project (EU-IST-FP6-26979), and the European ReSIST network of excellence (EU-IST-FP6-26764).

and a *platform* for its experimental evaluation. This architecture is exemplified by implementing a Distributed Black-Box (DBB) application which provides a virtual device, whose semantics is similar to avionics black-boxes, that track cars' history in a way that can be replayed in the event of a car accident. This application ensures information is securely stored using replication mechanisms, by means of exchanging positions between cars. Our implementation is based on three original services: a *Proximity Map*, a *Trust and Cooperation Oracle*, and a *Cooperative Data Backup.*

This DBB application is a good illustration of the use of the various middleware services and applications that users can benefit from thanks to mobile communicating devices, such as in the automobile context with car-to-car communication. As a "classical" black-box, its aim is to record critical data, such as engine / vehicle speed, brake status, throttle position, or the state of the driver's seat belt switch. As a "smart" black-box, it can also be used for extending the recorded information with contextual information concerning the neighboring vehicles, and ideally the various vehicles that were involved in a given accident. Indeed, information stored by the application leverages vehicle-based parameters and communication-induced information.

The proposed architecture is based on four main middleware building blocks, namely a Networking service, a Proximity Map, a Trust and Cooperation Oracle, and a Cooperative Data Backup service. This architecture and the DBB application will be described in Section 2. This distributed architecture being targeted to mobile nodes (automobiles), it has been implemented and evaluated on top of a mobile robot platform described in section 3. Finally, we conclude this paper and give some further trails of research in Section 4.

## 2 Architecture and system model

The work presented in this section was conducted in the course of the Hidenets project. Hidenets (HIghly DEpendable ip-based NETworks and Services) was a specific targeted research project funded by the European Union under the Information Society Sixth Framework Programme. The aim of Hidenets was to develop and analyze end-to-end resilience solutions for distributed applications and mobility-aware services in ubiquitous communication scenarios.

The overall architecture used in this work is depicted on Fig. 1. The mobile platform, the hardware and other experimental settings will be described later in Section 3. This architecture is a partial implementation of the Hidenets architecture and has been detailed in the projects deliverables, see e.g. [2] or [3]. Apart from standard hardware-related services (networking, localization...), we propose three new building blocks, targeted for mobile systems, that are described in the following subsections. The rationale of these building blocks is as follows:

*Proximity map.* Before being able to backup data, a mobile node has first to discover its neighbors and the resources and services they offer. The proximity

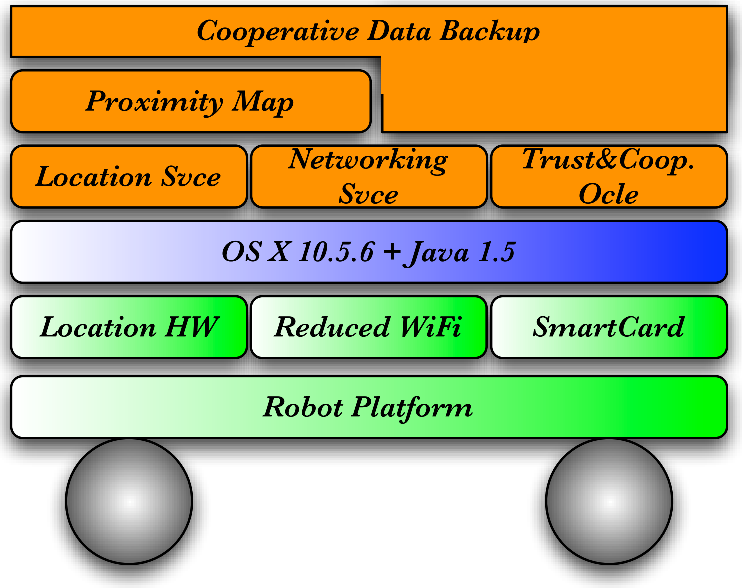

**Fig. 1.** Overall Architecture

map represents the local knowledge a node has about its vicinity. This can vary according to wideness (the number of communication hops represented on the map) and according to accuracy (how often is the map updated). Notice that, in this work, the aim of the proximity map is twofold: it is used to know which nodes can be used for cooperation, and is also used as a source of data to be backed up by the distributed black-box application, as we shall see later.

*Trust and cooperation oracle.* In order to interact with a priori unknown neighbors for critical services (e.g., collaborative backup), a node has to evaluate the level of trust it can assign to each of its neighbors. The purpose of the trust and cooperation oracle is to evaluate this level of trust and to incite nodes to cooperate with one another.

*Cooperative data backup.* The provision of a cooperative backup service at the middleware level is the major contribution of the architecture described in this paper. This service acts as a peer-to-peer storage resource sharing service for backup and restoration of critical data. There are four main tasks to achieve when considering cooperative data backup : (1) discovering storage resources in the vicinity, (2) negotiating a contract with the neighboring cars for the use of their resources, (3) handling a set of data chunks to backup and assigning these chunks to the negotiated resources according to some data encoding scheme and with respect to desired properties like dependability, privacy, confidentiality, and finally (4) taking care of the recovery phase, i.e., the data restoration algorithm.

```
// A node produces, sends and receives instances of Packet
class Packet {
   // size of payload is bounded
   protected byte[] payload;
   protected Node destinationNode;
   protected Node sourceNode;
   // Packets are typed, i.e., a client can request to receive packets of type "PMAP"
   protected String typeOfPacket;
   // There are actually several versions of the constructor
   Packet(byte[] message, Node destinationNode,  Node sourceNode,  String typeOfPacket);
}

class NetworkService {
   // NetworkService implements the singleton pattern,
   // i.e. only one instance is active
   // There is thus no constructor for NetworkService,
   // getInstance returns a reference to the unique instance of the NetworkService
   NetworkService getInstance();
   void basicBroadcast(Packet msg); // Performs a UDP broadcast of msg
   boolean unicast(Packet msg); // Performs a TCP unicast of msg
   Packet receive(String messageType);
       // Non-blocking receive operation,
       // if no message of type messageType in the queue, it returns null
}
```

**Fig. 2.** Adhoc Networking API

### 2.1 Communication and network layer

Since Java provides no specific support for ad hoc networking[1], we implemented a specific package for handling multiple WiFi interfaces. This package supports both UDP broadcasting and TCP unicasting. It handles indexing, choping and unchoping of arbitrary size messages and deals with typed messages. As we are only interested in local interactions within an entity's neighborhood, our network layer implements one-hop interactions only, and does not address the problem of routing in an ad-hoc network.

The package API is provided in Fig. 2. There are basically three important methods: `basicBroadcast`, `unicast` and `receive`. It is worth noticing that messages are typed by a string. This is very useful when several services can concurrently access the network service. They can use one or several types each and this way, won't consume other services messages. For example, the Proximity Map and the Trust and Cooperation Oracle, both described in the following sections, produce messages with, respectively, the `PMAP` and the `TCO` types. When

[1] There was a working group concerned with adhoc networking for Java. They produced a preliminary draft entitled "JSR-259 Ad Hoc Networking API" in 2006, but this draft didn't evolve since and no actual implementation was produced.

the Proximity Map calls the `receive` method, it receives messages of type `PMAP` only, even if there are `TCO`-typed messages in the queue. Those `TCO` typed messages thus remain in the queue until the Trust and Cooperation Oracle calls the `receive` method requesting a `TCO`-type message.

The rationale of having both broadcasting and unicasting available is simple. In a cooperative ad-hoc network, nodes need to interact with their neighbors which they do not a priori know. They can thus broadcast services advertisement or discovery requests in order to explore their vicinity. Once they acquired enough knowledge about the resources available in their network vicinity, they can use point-to-point communication, i.e. unicast, to access those resources. We believe that this is the only way cooperative services can be implemented on top of mobile adhoc networks and deal with nodes mobility and failures.

### 2.2 Localization and Proximity Map

In many applications dedicated to mobile nodes, and especially for cooperation-based applications, a node needs to interact with its neighbors. Furthermore, the quality of service that may be provided by a given component can vary according to the vicinity, e.g. the quantity of neighbors, their density, etc. It is then necessary to formalize this view of the vicinity into a more abstract representation. This is the purpose of the *Proximity Map* building block, that provides an abstraction of the physically attainable network of entities. The aim of this building block is to provide applications with information aggregated from both localization and networking layers.

Indeed, the goal of the proximity map is to gather physical information about nodes in the vicinity. When using its proximity map, a given node has a view of the nodes in its vicinity (defined as being the nodes which are reachable within $H$ hops), their location information, and the freshness of the pieces of information.

This problem has similarities with neighbor discovery protocols for ad-hoc routing algorithms, that can be divided into proactive schemes and reactive schemes. In a reactive scheme, information about routing is constructed on demand, i.e., as soon as a message has to be sent to a previously unknown destination. In a proactive scheme, the entity periodically sends messages on the network to look for new neighbors, and to verify the availability and reachability of already discovered neighbors. Since we are only interested in local interactions, and due to the fact that the set of entities is large and unknown to participants, we designed the proximity map as a proactive service.

Intuitively each node periodically beacons its proximity map to its 1-hop neighbors, and collects similar information from its direct neighbors. When merging these pieces of information, it is able to update its proximity map with new nodes that appeared as neighbors of neighbors, nodes which have moved, nodes whose connectivity changed, etc. The preliminary ideas about the proximity map can be found in [4]. To implement the proximity map, we use location-stamped beacons[2]. Each node keeps a map of its knowledge of the location and connec-

[2] In addition to the location of the node, beacons could also include other useful information about the sending node's radio coverage, its battery life, etc.

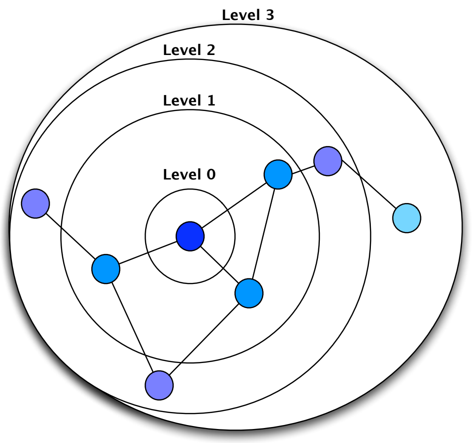

**Fig. 3.** The Proximity Map Knowledge

tivity of other nodes, which is represented as a graph as shown in Fig. 3. This graph is regularly updated when the node receives a beacon and is also regularly sent to the node's neighbors in its beacons.

*Map construction:* The pseudo-code of the Proximity Map protocol is given in Fig. 4. Here is the intuition behind it :

1. First, the node only knows its own location, it creates a proximity map `knowledge` and set itself as the root element of the graph.
2. The protocol is run every $D$ time units, it thus loops and sleeps so as to emulate a periodic thread.
3. It updates its position and broadcasts its knowledge to neighbors,
4. Then, the node receives its neighbors' Proximity Maps, and fusions them with its own.
5. After collecting all its neighbors information, it prunes its Proximity Map to a maximum of $H$ hops.
6. Then it performs the failure detection, i.e. removes nodes that are too old.

*Map accuracy:* Map accuracy can be evaluated with respect to two criteria:

- *timing*: is the map synchronized with reality or does it carry old connectivity and positioning information?
- *failure*: are the nodes present in a Proximity Map still alive and connected?

Timing issues are quite simple. First let's say that we call level 0 information about the node itself, level 1 information about its 1 hop neighbors, and level $n$ information about its $n$ hops neighbors. Then, quite intuitively, we can say that if beacons are sent every $D$ time units, then level $L$ information is $(L \times D)$ time units old. Because high-level knowledge is older and because having the

```
// First we initialize knowledge
knowledge = new ProximityMapStorage();
// At the beginning, we only know our own position
knowledge.setRootElement( myPosition );

// Loop forever
do {
  // Sleep D time units
  sleep(D);

  // Update our own position
  knowledge.changeRootElement( myPosition );

  //Send our PMAP to neighbors
  basicBroadcast(PMAP);

  // And also to receive the PMAPS and to integrate them
  while (there are PMAPs to be handled) {
    ProximityMap fromNeighbor = NetworkService.receivePMAP();
    if (fromNeighbor is not from myself) {

      // Extract information and add it to my own PMAP
      knowledge.addToPMAP(fromNeighbor);
    }
  }

  // Prune my PMAP so that it longest path does not exceed H
  knowledge.prune(H);

  // Perform failure detection, i.e. remove nodes whose information is too old
  for each node in knowledge {
    if ( (node.timestamp - currentTime) > failureDeltaTime ) {
      remove node from knowledge;
}}} while(true);
```

**Fig. 4.** The Proximity Map Algorithm

knowledge of the whole network is unnecessary, the maximum level of knowledge is bounded. The Proximity Map algorithm is pro-active and enables a node to know the other nodes physically present within the area if $H$ is sufficiently large. When $H$ is not large enough, or the coverage obtained is not sufficient, a reactive protocol may be used to complete location information further than $H$ hops.

In the absence of fault, at any time after $(H \times D)$ time units, a node knows the location and coverage of its $H$ hops neighbors. It should however be noted that at any time $t$, every node in the network has a different view of the connectivity since its level 1 information is $D$ time units old, its level $n$ is $n \times D$ time units old, etc.

At the moment, the bound $H$ is determined statically according to the application, the density of the network and other environment parameters. An interesting future work would be to modify this protocol to have a dynamic $H$ parameter that can be statically initialized and adapted at runtime according to the environment variation and to the application needs. It is worth noting that increasing $H$ has important consequences on both the network load and on the memory footprint of the proximity map. However, in some application scenarios, for example when the network density is low and when node connections get low, it would be quite beneficial to have a deeper knowledge of the network. An idea worth experimenting would be to adjust $H$ while keeping the memory footprint constant, i.e. exploring the network while there is some free memory for the protocol and garbage collecting the older nodes when freeing memory is necessary.

Regarding faults and failures, it is interesting to notice that the Proximity Map protocol is intrinsically resilient to faults. Indeed, let us first consider *crash failures*: when a node fails, it stops sending beacons and thus it will be removed from its neighbors' Proximity Maps after they didn't receive information about it for $failureDeltaTime$ time units. So a node that is $n$ hops from a failed node (with $n < H$)will remove the failed node from its map after $failureDeltaTime + n \times D$. Now, if we consider *intermittent faults*, for example a bad position given by the GPS device or even a proximity map containing bad information, since beacons are sent every $D$ time units, the faulty information will vanish and be replaced by the fresh and correct information. If we suppose that the faulty beacon is sent only once (i.e. the duration of the intermittent fault is less than $D$ time units), the effects on the global system will last at most $min(H \times D, failureDeltaTime)$: either the furthest node stays at $H$ hops and will receive fresh information after $H \times D$, or it will come closer and will get it sooner or even it will go further and henceforth never receive fresh information but will remove the failed node from its Proximity Map after $failureDeltaTime$ time units.

When building the Proximity Map, and in general in our whole architecture, we disregarded Byzantine behaviors because they should be avoided by the next building block: the Trust and Cooperation Oracle. However, it is interesting to note that given the intrinsic redundancy of the positioning and network connectivity among the various nodes of the network, it is relatively easy to imagine a byzantine-resilient version of the Proximity Map. Indeed, when a node $i$ receives a beacon from a node $j$, $i$ can check if the position advertised by $j$ is consistent with the views of its neighbors. Of course, this holds only when there is no collusion between $i$'s neighbors. However, this Byzantine-resilient version of the Proximity Map is future work, in the next section we present and discuss another approach for dealing with non cooperative, rationale, but also Byzantine nodes.

## 2.3 Trust and Cooperation Oracle

The trust and cooperation oracle (TCO) is our second building block for cooperative services. A cooperative service emerges from the cooperation of entities that are generally unknown to one another. Therefore, these entities have no a priori trust relationship and may thus be reluctant to cooperate. In cooperative systems without cooperation incentives, entities tend to behave in a rational way in order to maximize their own benefit from the system. The goal of the trust and cooperation oracle is therefore to evaluate locally the level of trust of neighboring entities and to manage cooperation incentives [5].

```
class TCO {
  // TCO implements the singleton pattern,
  // i.e. only one instance is active
  // There is thus no constructor for TCO,
  // getInstance returns a reference to the unique instance of the TCO
  TCO getInstance();

  // Main method for TCO, it returns the trust the nodes have in another node
  float trustLevel(NodeID n);
}
```

**Fig. 5.** Trust and Cooperation Oracle API

Synergy is the desired positive effect of cooperation, i.e., that the accrued benefits are greater than the sum of the benefits that could be achieved without cooperation. However synergy can only be achieved if nodes do cooperate rather than pursuing some individual short-term strategy, i.e. being rational[3]. Therefore, cooperative systems need to have cooperation incentives and rationality disincentives. There are several approaches to this, some are based on micro-economy and some others are based on trust. Typically, for micro-economic approaches, a node has to spend "money" for using a service and earns "money" for servicing other nodes. Regarding trust, a common approach is to use the notion of reputation, a level representing the level of trust that may be placed on a node, which can be computed locally by a single node, or collectively and transitively by a set of nodes. Another approach based on the notion of trust relies on the use of trusted hardware, e.g. a smart-card. Whatever the most appropriate approach in a given context, the TCO leverages this information by providing a single interface with simple semantics. Given a node identifier $n$, it returns the probability that this node $n$ cooperates correctly for the next interaction, as shown in Fig. 5.

When the various entities participating in a cooperative service belong to the same administrative domain, or to a limited number of domains, the question of trust establishment can be answered in a simple manner. For example, if we consider the case of a single administrative domain such as an enterprise, we can make the assumption that any node within the enterprise is going to

[3] A rational node always tries to maximize its benefits from the system, i.e. it behaves selfishly.

cooperate. The problem of the trust establishment is thus reduced to the question of identifying the nodes which are part of the enterprise. When multiple, but limited, administrative domains are involved, the question can sometimes be simplified in a similar manner.

In an automotive context, we consider that there are a limited number of different middleware providers. We can also state that it is at least unusual and potentially dangerous for vehicle owners to modify the software their vehicle is running, and that software updates are relatively rare. As a result, there are only a few different legacy middleware versions. We can thus consider that the middleware is certified, i.e., a trusted authority within the infrastructure domain can generate and distribute certificates. These certificates can be verified in the ad-hoc domain by a trusted hardware, e.g. in the Hidenets platform a smart-card. In this setting the concept of trust can be seen as all-or-nothing: when the certificate is verified, the middleware is legitimate, full trust is granted; on the other hand, when the certificate cannot be verified, this means that the middleware was modified and henceforth no trust can be given to the node.

In other application domains, such as cooperative backup of personal data stored on mobile devices such as smartphones for example, such a black or white notion of trust is not acceptable. Indeed, in such open settings, where many heterogeneous hardware and software cohabit, trust should rather be established based on the behavior of the nodes and their users than on the middleware legacy. The interested reader will find several approaches to deal with trust establishment, such as in [6] or in [5]. These different approaches can obviously be used to realize the Trust and Cooperation Oracle API and return a more balanced vision of trust, i.e. varying between 0 and 1.

## 2.4 Cooperative Data Backup service

The cooperative backup service aims to improve the dependability of data stored by participating nodes by providing them with mechanisms to tolerate hardware or software faults, including permanent faults such as loss, theft, or physical damage. To tolerate permanent faults, the service must provide mechanisms to store the users' data on alternate storage nodes using the available communication means. The problem of cooperative backup of critical data can be divided in three steps: *i*) discovering storage resources in the vicinity (this step is performed using the proximity map service), *ii*) negotiating a contract with the neighboring nodes for the use of their resources (this step uses the trust and co-operation oracle), and *iii*) handling a set of data chunks to backup and assigning these chunks to the negotiated resources according to a data encoding scheme and with respect to desired properties of dependability, privacy, availability and confidentiality. The service is also in charge of the recovery phase, i.e., the data restoration algorithm.

The Cooperative Data Backup service provision is designed using the following principles:

- A client of the service provides a data stream to be backed up to the `backup` operation with a unique identifier for the stream.

```
void run() { // This is the main loop method
  loop { // It serves client requests and processes network messages
    sleep(D); // Sleeps D time units
    handleRestoration(); // Handles client restoration requests
    receiveCBPackets(); // Processes CooperativeBackup messages
    backupData(); // And processes client backup requests
}}
void handleRestoration() {
  for all blockID in missingBlocks {
    // We first try to find all locally-available requests
    if blockID is found locally {
      store the corresponding block in replyStorage;
      remove blockID from missingBlocks
    } else { // if the block has not been found locally then send a request for it
      Net.basicBroadcast(request for blockID, "CooperativeBackup");
}}}
void receiveCBPackets() {
  while there are packets typed "CooperativeBackup" to be handled {
    if source of packet is trusted enough by the TCO {
      if packet is a storage request {
        unpack the block contained in p and store it locally under blockID
      } else if packet p is a restoration request {
        unpack the blockID requested in p
        search in the local storage for blockID
        if the block was found, unicast it to the sender of the request
      } else { // it is then a reply to a restoration requested
        unpack the blockID and the block data sent in p
        store block locally with blockID
        remove blockID from missingBlocks
}}}}
void backupData() {
  // Example: straightforward backup policy: each block is distributed only once
  Build the list of nodes candidates for backup, using ProximityMap and TCO
  while (there are blocks to backup in backupStore and there are candidates) {
    for each candidate taken in a randomized order {
      unicast a storage request for the next block in backupStore to the candidate
      if the unicast was successful, remove the block from the backupStore;
    }
    rebuild the candidates list
}}
```

**Fig. 6.** Cooperative Data Backup Algorithm

– The stream passes through a series of chopping and indexing operations in order to produce a set of small (meta-) data chunks to be backed up (more details can be found in [7]).
– A backup thread runs periodically, it processes the block buffer, queries the Proximity Map service and the Trust and Cooperation Oracle in order to

produce a potential contributors list. Then it places data blocks on contributors according to given placement and replication strategies, as described in [7] and [8].

When the client wants to restore data, it can either submit the unique identifier of the stream to the asynchronous restore operation and then poll it periodically, or it can directly call the synchronous restore operation that will return when the data has been successfully restored. To that means, a periodic thread handles the restoration waiting queue: it looks for given IDs, unpacks the received blocks and potentially adds new identifiers to the waiting queue according to the decoding operation on received data chunks (i.e. data or meta-data).

Fig. 6 gives the pseudo-code for both cooperative backup. A periodic thread handles restoration requests, then processes network packets and then handles backup requests. Restoration first looks locally if the requested block is found, if it is not the case it broadcasts a request looking for the block (by sending its unique block ID). Processing network messages involves dealing with three type of messages: 1-Storage requests, 2-Restoration requests and 3-Restoration replies. Of course only the messages sent by trusted sources (by the TCO) are processed. Handling backup requests implies implementing a particular backup strategy, i.e. placing each block a certain number of times on the nodes available in the vicinity (and trusted by the TCO).

In [8], we discuss various backup strategies and we provide an analytical evaluation of the storage cost and dependability of these strategies as a function of a few parameters such as: $\alpha$ the rate of peers' encounters, $\beta$ the rate of infrastructure connectivity (to perform backup using the Internet), and of course $\lambda$ the failure rate. Basically, we showed that the cooperative backup approach is beneficial (i.e., yields data dependability an order of magnitude higher than without cooperative backup) only when $\frac{\beta}{\lambda} > 2$ and $\frac{\alpha}{\beta} > 10$. We demonstrated that cooperative backup can decrease the probability of data loss by a factor that can be as large as the ad hoc to Internet connectivity ratio $\frac{\alpha}{\beta}$.

The Cooperative Data Backup Service API is given in Fig. 7. This service is very simple to use. First, the client of the service needs to get an instance of the service through a call to getInstance. Then it can send backup and restoration requests through *submitBackupData* and *submitRestore* respectively. The client can get the restored data back, later on, using the *restore* method. This method, as the other two methods, is non-blocking and returns *null* if the data is not restored yet. With the Cooperative Data Backup Service, data is identified through a unique identifier (we used the Java's UUID in the implementation). This means that clients of the service have to compute (and remember) these identifiers. This is classically obtained by constructing a directory containing all the identifiers of the backed up data and by backing up this directory using a static identifier, that one can easily remember, such as the node MAC address for example.

```
class CooperativeBackup {
  // CooperativeBackup implements the singleton pattern,
  // i.e. only one instance is active
  // There is thus no constructor for CooperativeBackup,
  // getInstance returns a reference to the unique instance of the CooperativeBackup
  CooperativeBackup getInstance();

  // submitBackupData feeds the CooperativeBackup with data to be backed up
  //  the data can be later restored by providing its unique identifier UUID
  public void submitBackupData(byte data[], UUID key );

  // submitRestore requests the restoration of the data identified by uid
  // the data will be available later through a call to restore
  public void submitRestore(UUID uid);

  // restore returns the data identified by uid if the data has been restored
  //  and returns null otherwise
  public byte[] restore(UUID uid);
}
```

**Fig. 7.** Cooperative Data Backup API

## 2.5 The Distributed Black-Box application

Using the above described services, we implemented a Distributed Black-Box application. In a few words, this application backs up a stream of data for every car that consists of a periodic sampling of a car's proximity map. The cooperative backup service is used to replicate these streams among neighboring cars, or to an infrastructure when connectivity permits it. The stream of any participant (be it crashed or not) can then be restored either from neighboring devices (cars in ad-hoc mode), or from the infrastructure. In more details, the application maintains a Black-box view of the car vicinity. This view contains the car position, its proximity map pruned to a maximum of 10 hops, a timestamp. This view is serialized and submitted to the Cooperative Data Backup Service using a unique identifier that consists in the concatenation of the car license plate number and the current time. When a car crashes and its Black-box view needs to be reconstructed, one has to submit the crashed car license plate number and the time interval in which one is interested. For example, if an accident happened to a specific car on the 1st of july around noon, one can want to get the Black-box views of the car “02 ARUM 31”, between “1st July 2010 - 12:05” and “1st July 2010 - 12:15”. The Distributed Black-Box application will try to restore all the Black-box views corresponding to this pattern. After these requests propagate through the network, and when the corresponding replies begin to arrive, the restoration client can travel back in time and try to understand what happened to the crashed car.

## 3 The ARUM experimental platform

To the best of our knowledge, little research has been done on the evaluation of resilience in ubiquitous systems. Most of the literature in this domain concerns evaluation of users experience and human-computer interfaces. However, some work is also looking at defining appropriate metrics for the evaluation of distributed applications running on ubiquitous systems [9, 10]. [11] is looking at a general approach to evaluate ubiquitous systems. In this paper, the authors argue that quantitative measurements should be complemented with qualitative evaluation. Their argument is that there is a number of problems for which evaluation cannot be easily quantified. Thus an evaluation should be conducted using an hybrid quantitative/qualitative strategy.

It is clear that the area of resilient computing has proposed a number of contributions concerning the evaluation of distributed systems and this paper will not survey this domain. Analytical evaluation is probably the most popular technique, such as within Assert [12] in the avionics application domain. More recently, experimental evaluation started to gain attention. The approach taken is often based on dependability benchmarking, for example DBench [13] addresses dependability benchmarking of operating systems.

In the ubiquitous and mobile computing area, evaluation of resilient mechanisms remains an open problem. In most cases, the proposed algorithms are evaluated and validated using network simulators [14, 15]. Since simulators use a model of physical components, such as wireless network cards and location systems, this raises concerns on the coverage of the assumptions that underlie the simulation [16]. Little work concerning the evaluation of algorithms in a realistic mobile experimental environment is available. However, related work is discussed in Section 3.4.

### 3.1 Scalability.

The above mentioned lack of a realistic mobile experimental environment calls for the development of a realistic platform, at a laboratory scale, to evaluate and validate fault-tolerance algorithms (in particular the services described in Section 2) targeting systems comprising a large number of communicating mobile devices equipped with various sensors and actuators. The goal is to have an experimentation platform allowing for reproducible experiments (including mobility aspects) that will complement validation through simulation. As we will see, an important issue within this platform is related to changes of scale so as to emulate many various systems.

In the ARUM[4] project, we are developing an experimental evaluation platform composed of both fixed and mobile devices [17–19]. Technically speaking, each mobile device is composed of some programmable mobile hardware able to

[4] ARUM stands for an Approach for the Resilience of Ubiquitous Mobile systems. It is an internal project funded by the Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS).

carry the device itself, a lightweight processing unit equipped with one or several wireless network interfaces and a positioning device. The fixed counterpart of the platform contains the corresponding fixed infrastructure: an indoor positioning system, wireless communication support, as well as some fixed servers. Our platform is set up in a room of approximately $100m^2$ where mobile devices can move around. By changing scale, we can emulate systems of different sizes. Hardware modeling of this type of system requires a reduction or increase of scale to be able to conduct experiments within the laboratory. To obtain a realistic environment, all services must be modified according to the same scale factor.

For example, if we consider a Vehicular Ad-hoc NETwork (VANET) experiment, a typical GPS in a moving car is accurate to within $5-20m$. So, for our $100m^2$ indoor environment to be a scaled down representation of a $250000m^2$ outdoor environment (that represent a scale reduction factor of 50 for distances), the indoor positioning accuracy needs to be at least $10-40cm$. The following table summarizes the required change in scale for all peripherals of a node.

| **Device** | **Real Accuracy** | **Scaled Accuracy** |
|---|---|---|
| Wireless | range: $100m$ | range: $2m$ |
| GPS | $5m$ | $10cm$ |
| Node size | a few meters | a few decimeters |
| Node speed | a few $m/s$ | $< 1m/s$ |

### 3.2 Technological aspects.

**Positioning** Several technologies are currently available for indoor location [20], mostly based either on scene analysis (e.g. using motion capture systems) or on triangularization (of RF and ultrasound [21] or wireless communication interfaces [22]). In this section, we describe the various systems we used, and analyze both their positive and negative aspects.

To reach our desired level of accuracy for indoor positioning, we first used a dedicated motion capture technology that tracks objects based on real-time analysis of images captured by infra-red cameras. The Cortex[5] system is able to localize objects at the millimeter scale, which is more than enough for the VANET setting. This technology uses a set of infrared cameras, placed around the room, that track infrared-visible tags. All cameras are connected to a server that computes, based on all cameras images, the position of every tag in the system. We equipped our small robots with such tags, and the computers on the robots connect to the server to get their positioning information. Although the precision attained was more than enough for our needs, the system has some drawbacks: the whole system is very expensive (in the order of $100k$€), calibration is a tedious task, and infrared signals cannot cross obstacles such as humans.

To overcome these limitations, we are currently developing a new localization system, based on two different technologies that have complementary advantages. The first one is based on infrared cameras, as for Cortex, but the

[5] http://www.motionanalysis.com

system is reversed: cameras are on-board, and locate themselves by tracking statically placed infrared-visible tags. This system is coupled with an Ultra-Wide-Band-based localization system, Ubisense. Ultra-Wide-Band-based localization (UWB) is performed by 4 sensors, placed in the room at each corner, that listen for signals sent by small tags that emit impulses in a wide spectrum. Such impulses can traverse human bodies and small obstacles, so the whole system is robust to external perturbation, but, from our preliminary measurements, attainable precision is less than $10cm$.

We thus advocate that the coupling of these two technologies will result in a localization system with desirable properties: it is relatively cheap, it is robust to external perturbations such as obstacles, and has most of the time a precision about the order of a centimeter.

**Mobility** Another important question is how to make the devices actually mobile. Obviously, when conducting experiments, a human operator cannot be behind each device, so mobility has to be automated. This is why we considered the use of simple small robot platforms in order to carry around the platform devices. The task of these robots is to "implement" the mobility of the nodes. The carried devices communicate with the robot through a serial port. This way they can control the mobility, i.e. the trajectory, the stops and continuations, the fault-injection, etc.

A node in the system is implemented using a laptop computer, that includes all hardware devices and the software under testing, that is carried by a simple robotic platform, the Lynxmotion 4WD rover. A 4WD rover is able to carry a full node during a few hours, running at a maximum speed of $1m.s^{-1}$, which is consistent with our assumptions.

To have reproducible patterns of mobility, the rover embarks a dedicated software that moves the robot using two different schemes. Both designs allow for testing different algorithms using the same mobility pattern, and for testing the same algorithm with different mobility scenarios.

In the simple scheme, a robot is following a black line on the floor. This solution is easy to implement but imposes that the operator "draws" the circuit for every different mobility pattern.

The second scheme couples a predefined mobility pattern with the positioning service and ensures a given node moves according to the predefined pattern, programmed by the operator. This solution is more flexible: each node has its own mobility pattern specified for each experiment.

**Communication** The last and most important design issue for the platform concerns wireless communications. Indeed, the communication range of the participants (mobile nodes and infrastructure access-points) has to be scaled down according to the experiment being conducted. For example, with a VANET experiment, a typical automobile has a wireless communication range of a few hundred meters, say $200m$. With a scale reduction factor fixed at 50, the mobile devices communication range has to be limited to $4m$. However, to cope with other experiments and other scale reduction factors, this communication range should ideally be variable.

A satisfying solution consists in using, for this purpose, signal attenuators placed between the WiFi network interfaces and their antennas. An attenuator is an electronic device that reduces the amplitude or power of a signal without appreciably distorting its waveform. Attenuators are passive devices made from resistors. The degree of attenuation may be fixed, continuously adjustable, or incrementally adjustable. In our case, the attenuators are used to reduce the signal received by the network interface. The necessary capacity of the attenuators depends on many parameters such as the power of the WiFi interfaces and the efficiency of the antennas, but also on the speed of the robot movements, the room environment, etc.

### 3.3 Application scenario.

As can be seen on Fig 1, the middleware described in this article is running on top of Apple OS X.5.6 and Java 1.5. The hardware (Macbook with additional WiFi interface and some localization hardware) is carried by a Lynxmotion 4WD rover. The resulting platform can be seen on Fig. 8. We currently own four fully equipped robots. We were thus able to emulate the Distributed Black-Box in a setting with three cooperating cars and a police coming after an accident has taken place. During the first part of the scenario, the three cars backup each others' Black-Box data for each other, then one of the cars looses control and leaves the circuit track to crash in a wall. After the accident has been reported, including the ID of the crashed car and the approximated time of the accident, the police enters the scene and requests restoration of the black box data for a given period of time that surrounds the accident. Once the data is successfully restored, the police is then able to replay the film of the accident, and to identify the other involved cars if there are any. A movie of this scenario is available at http://homepages.laas.fr/mroy/hidenets/.

### 3.4 Related work.

Relatively little work concerning the evaluation of algorithms in a realistic mobile experimental environment is available. Most of the available platforms are based on wired emulation of wireless networks [23]. Wired wireless emulators such as EMPOWER [24], and EMWIN [25] use a centralized emulation layer and rely on switching equipment to disseminate messages to "mobile" nodes. Non-centralized wireless testbed emulators such as SEAWIND [26] or SWOON [27] rely on a wired configurable testbed similar to Emulab [28]. These testbed emulators make use of various link shaping techniques to approximate a wireless link. Typically, a special node is used for one or more links that need to be emulated. The quality of the emulation can suffer since these testbeds utilize switching equipment and multiple nodes to propagate messages. Both Mobile Emulab [29] and MiNT [30] use robots to emulate mobility of wireless nodes. The mobile version of Emulab embarks Motes to emulate a wireless sensor network; wireless

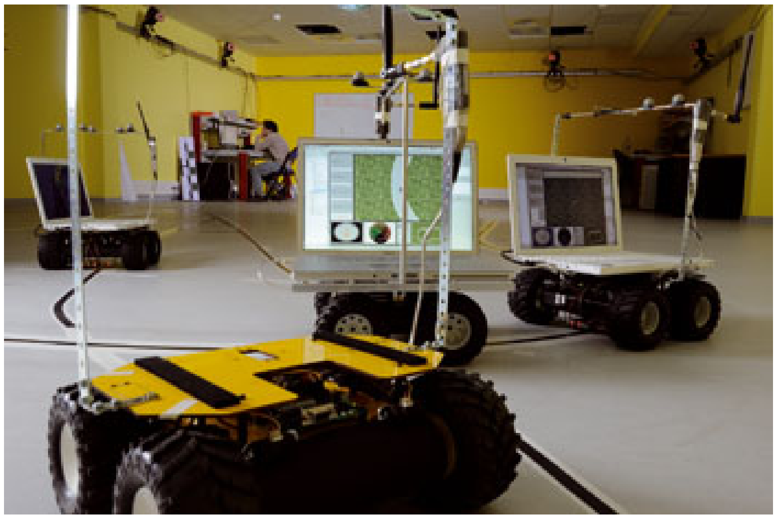

**Fig. 8.** The ARUM platform

experiments are carried at the building scale. Similarly to our platform, MiNT uses signal attenuators to reduce the space needed for experiments [31]. However, in order to reduce MiNT's node cost, the positioning subsystem is based on simple web-cams and henceforth is not precise.

### 3.5 Mobility Issues.

Building a platform for evaluating mobile systems was clearly a challenge, as illustrated by the small number of other available platforms that implement real mobility. But placing laptops on wheels is not enough to evaluate distributed mobile applications in meaningful mobile configurations. The way the nodes move, both from their own perspective, but also according to other nodes movement, is a very interesting scientific issue that needed to be addressed. We believe that the usual mobility models used for the evaluation of mobile systems are not satisfactory. A mobility model dictates how the nodes, once distributed in the space, move. A mobility model involves the nodes' locations, velocities and accelerations over time. The topology and movement of nodes are key factors in the performance of the system under study. Because the mobility of the nodes directly impacts the performance of the protocol, if the mobility model does not reflect realistically the environment under study, the result may not reflect the performance of the system in the reality. The majority of existing mobility models for ad hoc networks do not provide realistic movement scenarios [32]. We are currently working on the use of real mobility traces from various sources in order to build more realistic mobility models to use in our analytical and experimental evaluation. The production and usage of such real-life mobility traces also raise a lot of privacy concerns that we recently began to address, see e.g. [33].

## 4 Conclusion

In this paper, we presented a middleware for building resilient cooperative mobile systems. This middleware is based on our belief that in an ubiquitous environment, with many fixed and mobile communicating nodes that do not know each other a priori, local cooperation with neighboring nodes is the approach to follow in order to build fast and reliable applications and services. This calls for network and middleware layers that encourages the application to first discover available services in the vicinity, then evaluate trustiness of available resources, and finally interact with those services. This is the purpose and the philosophy behind the design of the Network and communication layer, the Proximity Map, and the Trust and Cooperation Oracle. Another important aspect of ubiquitous and mobile computing models is the fact that communication can be ephemeral. This is the reason why we designed a Cooperative Data Backup Service: a node can leverage the ephemeral encounters it makes in order to replicate and disseminate its critical data. We have shown that using these building blocks, it is very easy to build a resilient cooperative application: a Distributed Black-box that reliably stores critical data.

Up to now, most of the algorithms and protocols for mobile systems were evaluated using simulators and analytical techniques in the best cases. We advocate that these evaluation techniques do not capture all the complexity inherent to a ubiquitous and mobile computing environments. Hence, we developed a platform based on (1) mobile robots, (2) scaled-down wireless communication interfaces, and (3) extremely precise localization, in order to perform realistic experimental evaluation of mobile services and applications.

We used with success this platform to run and evaluate the middleware building blocks, and the Distributed Black-box application presented in this paper. This raised a lot of technical questions but also many design issues. Indeed, the middleware building blocks were profoundly influenced by the fact that it was implemented for real-life use, using real (i.e. not simulated) hardware and that the whole hardware platform was really mobile. Both the middleware and the platform influenced each other in a kind of virtuous circle. Both of them will be reused in other settings, to build and to evaluate other resilient architectures and other mobile cooperative applications.

## References

1. M. Roy, F. Bonnet, L. Querzoni, S. Bonomi, M.-O. Killijian, and D. Powell, “Georegisters: An abstraction for spatial-based distributed computing,” in *Int. Conf. On Principles Of DIStributed computing (OPODIS), LNCS 5401*, 2008, pp. 534–537.
2. J. Arlat and M. Kaâniche(editors), “Hidenets. revised reference model. deliverable nr. d1.2,” LAAS-CNRS, Contract Report nr. 07456, September 2007.
3. A. Casimiro et al., “Resilient architecture (final version),” LAAS-CNRS, Tech. Rep. 08068, December 2008. [Online]. Available: http://www.di.fc.ul.pt/tech-reports/07-19.pdf

4. M.-O. Killijian, R. Cunningham, R. Meier, L. Mazare, and V. Cahill, "Towards group communication for mobile participants," in *Proceedings of Principles of Mobile Computing (POMC)*, 2001, pp. 75–82.
5. L. Courtès, M.-O. Killijian, and D. Powell, "Security rationale for a cooperative backup service for mobile devices," in *Proceedings of the Latin-American Symposium on Dependable Computing (LADC)*. Springer-Verlag, 2007, pp. 212–230.
6. O. Nouha and R. Yves, "Cooperation incentive schemes," Institut Eurecom, France, Tech. Rep. EURECOM+2026, 09 2006.
7. L. Courtes, M.-O. Killijian, and D. Powell, "Storage tradeoffs in a collaborative backup service for mobile devices," in *European Dependable Computing Conference (EDCC)*, 2006, pp. 129–138.
8. L. Courtes, O. Hamouda, M. Kaaniche, M.-O. Killijian, and D. Powell, "Dependability evaluation of cooperative backup strategies for mobile devices," in *Pacific Rim Dependable Computing*, 2007, pp. 139–146.
9. P. Basu, W. Ke, and T. D. C. Little, "Metrics for performance evaluation of distributed application execution in ubiquitous computing environments," *Workshop on Evaluation Methodologies for Ubiquitous Computing at Ubicomp'01*, 2001. [Online]. Available: http://zing.ncsl.nist.gov/ubicomp01/
10. P. Castro, A. Chen, T. Kremenek, and R. Muntz, "Evaluating distibuted query processing systems for ubiquitous computing," *Workshop on Evaluation Methodologies for Ubiquitous Computing at Ubicomp'01*, 2001. [Online]. Available: http://zing.ncsl.nist.gov/ubicomp01/
11. M. Burnett and C. P. Rainsford, "A hybrid evaluation approach for ubiquitous computing environments," *Workshop on Evaluation Methodologies for Ubiquitous Computing at Ubicomp'01*, 2001. [Online]. Available: http://zing.ncsl.nist.gov/ubicomp01/
12. J. Arlat, M. R. Barone, Y. Crouzet, J.-C. Fabre, M. Kaaniche, K. Kanoun, S. Mazzini, M. R. Nazzarelli, D. Powell, M. Roy, A. E. Rugina, and H. Waeselynck, "Dependability needs and preliminary solutions concerning evaluation, testing and wrapping," LAAS, Toulouse, Tech. Rep. 05424, 2005.
13. K. Kanoun, H. Madeira, F. Moreira, M. Cin, and J. Garcia, "Dbench - dependability benchmarking," in *Proc. of the Lecture Notes in Computer Science (LNCS), Springer-Verlag, Fifth European Dependable Computing Conference (EDCC-5)*, April 2005.
14. S. R. Das, R. Castañeda, and J. Yan, "Simulation-based performance evaluation of routing protocols for mobile ad hoc networks," in *Mob. Netw. Appl.*, vol. 5, no. 3. Hingham, MA, USA: Kluwer Academic Publishers, 2000, pp. 179–189.
15. E. B. Hamida, G. Chelius, and J. M. Gorce, "On the complexity of an accurate and precise performance evaluation of wireless networks using simulations," in *11th ACM-IEEE Int. Symp. on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, 2008.
16. D. Cavin, Y. Sasson, and A. Schiper, "On the accuracy of manet simulators," in *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*. New York, NY, USA: ACM Press, 2002, pp. 38–43.
17. M.-O. Killijian, N. Rivière, and M. Roy, "Experimental evaluation of resilience for ubiquitous mobile systems," in *Proc. of UbiComp, Workshop on Ubiquitous Systems Evaluation (USE)*, 2007, pp. 283–287.
18. M.-O. Killijian and M. Roy, "Brief announcement: a platform for experimenting with mobile algorithms in a laboratory," in *PODC*, S. Tirthapura and L. Alvisi, Eds. ACM, 2009, pp. 316–317.

19. M.-O. Killijian, M. Roy, G. Severac, and C. Zanon, "Data backup for mobile nodes : a cooperative middleware and experimentation platform," in *Proc. of the Workshop on Architecting Dependable Systems of the IEEE International Conference on Dependable Systems and Networks (DSN-2009), Lisboa, Portugal*, 2009.
20. J. Hightower and G. Borriello, "A survey and taxonomy of location systems for ubiquitous computing," 2001. [Online]. Available: citeseer.ist.psu.edu/hightower01survey.html
21. A. Smith, H. Balakrishnan, M. Goraczko, and N. B. Priyantha, "Tracking Moving Devices with the Cricket Location System," in *2nd International Conference on Mobile Systems, Applications and Services (Mobisys 2004)*, Boston, MA, June 2004.
22. N. S. Correal, S. Kyperountas, Q. Shi, and M. Welborn, "An uwb relative location system," in *Proc. of IEEE Conference on Ultra Wideband Systems and Technologies*, November 2003.
23. D. Havey, R. Chertov, and K. Almeroth, "Wired wireless broadcast emulation," in *5th International workshop on Wireless Network Measurements (WiNMee)*, 2009.
24. P. Zheng and L. M. Ni, "Empower: A network emulator for wireline and wireless networks," in *In Proceedings of IEEE InfoCom. IEEE Computer and Communications Societies*, 2003.
25. ——, "Emwin: Emulating a mobile wireless network using a wired network," in *In Proceedings of the 5th ACM international workshop on Wireless mobile multimedia.* ACM Press, 2002, pp. 64–71.
26. M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, and K. Raatikainen, "Seawind: a wireless network emulator," in *In Proceedings of 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, 2001.
27. Y. L. Huang, J. D. Tygar, H. Y. Lin, L. Y. Yeh, H. Y. Tsai, K. Sklower, S. P. Shieh, C. C. Wu, P. H. Lu, S. Y. Chien, Z. S. Lin, L. W. Hsu, C. W. Hsu, C. T. Hsu, Y. C. Wu, and M. S. Leong, "Swoon: a testbed for secure wireless overlay networks," in *CSET'08: Proceedings of the conference on Cyber security experimentation and test.* Berkeley, CA, USA: USENIX Association, 2008, pp. 1–6.
28. B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation.* Boston, MA: USENIX Association, Dec. 2002, pp. 255–270.
29. D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile emulab: A robotic wireless and sensor network testbed," in *INFOCOM.* IEEE, 2006.
30. T.-c. Chiueh, R. Krishnan, P. De, and J.-H. Chiang, "A networked robot system for wireless network emulation," in *RoboComm '07: Proceedings of the 1st international conference on Robot communication and coordination.* Piscataway, NJ, USA: IEEE Press, 2007, pp. 1–8.
31. P. De, A. Raniwala, S. Sharma, and T. cker Chiueh, "Mint: a miniaturized network testbed for mobile wireless research," in *INFOCOM.* IEEE, 2005, pp. 2731–2742.
32. M. Musolesi and C. Mascolo, "Mobility models for systems evaluation," in *State of the Art on Middleware for Network Eccentric and Mobile Applications (MINEMA). Springer*, February 2009.
33. S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez, "Gepeto: A geoprivacy-enhancing toolkit," in *AINA Workshops.* IEEE Computer Society, 2010, pp. 1071–1076.