A Framework for Pervasively Shared Situational Awareness in Mobile Ad Hoc Environments

By

Fernando J. Maymí Fernández

A dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY
In
COMPUTER AND INFORMATION SCIENCE AND ENGINEERING

University of Puerto Rico
Mayagüez Campus
2009

Approved by:

Ronald C. Dodge, Jr., Ph.D. Member, Graduate Committee	Date
Wolfgang A. Rolke, Ph.D. Member, Graduate Committee	Date
Jaime Seguel, Ph.D. Member, Graduate Committee	Date
Manuel Rodríguez Martínez, Ph.D. President, Graduate Committee	Date
Cristina Pomales García, Ph.D. Graduate School Representative	Date
Nestor Rodriguez Rivera, Ph.D. Chairperson of the Program	Date

ABSTRACT

A Framework for Pervasively Shared Situational Awareness in Mobile Ad Hoc Environments

By

Fernando J. Maymí Fernández

This dissertation presents a novel approach to sharing situational awareness among a group of geographically dispersed mobile networked devices. We divide our efforts into two parts: getting the information to the right places, and keeping it there. We start by introducing a stochastic approach to transporting geospatial messages that shows significantly better performance than the existing deterministic approaches, particularly in adverse network topologies. Under normal conditions, our model performs better than comparable existing approaches. Under pathological conditions, however, all other approaches fail while our model remains viable.

Next, we introduce a novel cooperative spatial-aware cache that can be thought of as residing in a region rather than in any given node. Each node that participates in the cache holds an expendable part of the data, so that the loss of any node or small group of nodes can be tolerated with little or no degradation of service. The results of both our analysis and simulations show that our approach works significantly better than local caching alone both in terms of hit-ratios as well as cache survivability. In fact our shared spatial caches perform one order of magnitude better than local caches in terms of cache hit ratios.

RESUMEN

Un Marco para Compartir Conocimiento Circumstancial Universalmente

Por

Fernando J. Maymí Fernández

Esta tesis presenta un acercamiento novedoso a el compartimiento de conocimiento circumstancial entre un grupo de dispositivos inalámbricos móbiles que estan geográficamente esparcidos. Dividimos nuestros esfuerzos en dos partes: lograr que la información llegue a los lugares correctos, y mantenerla ahí. Empezamos introduciendo un acercamiento estocástico al transporte de mensajes entre los dispositivos el cual demuestra un rendimiento bastante mejor que el de los acercamientos determinísticos actuales, particularmente en topologías adversas. Bajo condiciones normales, nuestro modelo demuestra un mejor rendimiento que el de los acercamientos actuales. Bajo condiciones patológicas, sin embargo, todos los demás acercamientos fracazan, mientras que el nuestro permanece factible.

Luego, introducimos un novedoso caché espacio-consciente y cooperativo el cual pudiera considerarse que existe en una región en vez de en un nodo dado. Cada nodo que participa en el caché mantiene una parte fungible de la información, de manera que la pérdida de un nodo o un pequeño grupo de nodos puede ser tolerada con ninguna o muy poca degradación del servicio. Los resultados de tanto nuestro análisis como de nuestras simulaciones demuestran que nuestro acercamiento funciona mucho mejor que un caché local tanto en términos de rendimiento como de supervivencia. De hecho, nuestros cachés espaciales compartidos dan un rendimiento que es un orden de magnitud mejor que los cachés locales.

To my wife... for everything

ACKNOWLEDGMENTS

Thanks to all those without whom this work would have never happened. This truly was a team effort entailing contributions from many wonderful professionals. The list is long, but some really need to be mentioned by name. To the rest, please forgive the omission. I would like to acknowledge the help of ...

- Andre Sayles, and Gene Ressler, who encouraged me to pursue a Ph.D. and got the Army to foot the bill,
- Paul Manz, who planted in my mind the seed from which this dissertation grew, and always ensured my work was adequately supported,
- Manuel Rodríguez Martínez, who kept me on track these last few years despite my best efforts to branch off into one of the many fascinating issues I discovered along the way,
- Wolfgang Rolke, who showed me just how little I know of probability and statistics, and helped me fix that,
- Ronald Dodge, who has always been an inexhaustible source of support, whether it's been ideas, a plane ticket, or a document mailed overnight,
- Eliana Valenzuela, who showed me how to navigate the dizzying mazes of academic bureaucracies, and
- my mother, Emma Fernández, who gave me the conviction that I can accomplish anything I set my mind to.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Our survival as a species has always depended on our ability to acquire information about our immediate area and share that information with our neighbors. This dependency is best illustrated by the situations faced by our emergency response personnel in times of crisis. Whether it is a firefighter who needs to know if the building he is searching is about to collapse, or a police officer being aware that the riot shes helping control is surrounding her, the need for situational awareness (SA) in crisis management is prevalent.

Even as our command and control centers benefit from the latest advances in technology, it is the last mile, that link to the person on the ground, which still relies on the limited effectiveness of hand-held radios for those selected ones that get them. Imagine a member of a search and rescue team looking for survivors in the aftermath of a natural disaster. She may have a hand-held radio and she may even be within range of someone else in her team. Still, neither she nor her teammates may be aware that, miles away, a weakened dam just burst, releasing a torrent of water that is heading for the team. Even if every person had a working radio, could they be warned in time? Would anyone even know that the team was in harms way?

We developed a prototypical military information system that warns dismounted personnel of incoming indirect fire attacks while reporting their location to existing command and control systems. The main component of this system is a pager-like device worn by each

person. This device periodically transmits their location and listens for threat warnings. A typical usage scenario is depicted in Fig. 1.1. When an insurgent launches a mortar at friendly forces (a), friendly counter-mortar radars pick up the round before it even reaches its apogee (b). This event is automatically added to command and control systems (c), which interface with our system to wirelessly transmit the event information (i.e. the inbound mortar round) to all soldiers within the predicted blast area (d). When the event notification is received, the pager determines whether or not it is inside the affected area and, if so, sounds an audible or vibratory alarm that increases in frequency as the predicted time of impact approaches. Depending on the type of ammunition and its trajectory, personnel may have a few seconds to well over a minute to improve their protective posture before enemy rounds detonate near them [36].

This thesis builds upon this prototypical architecture so that it enables the right information to be available at the right place and time to the right users in a variety of scenarios. This architecture relates information to users and, based on these relationships, places spatial and temporal constraints on the system-wide information exchanges. Furthermore, the information within the architecture is tagged with metadata that enables the participants of the network to determine its relative confidence level. We call this architecture Ancile.

1.1 Motivation

Ancile was born of the need to warn dismounted troops in Iraq and Afghanistan of the threat of incoming mortar rounds. We developed a system of pagers and bridging devices that worked very well during live-fire testing in the deserts of Arizona. During these tests, mortar rounds were fired at a target area, picked up by radars, and a warning message transmitted across multiple networks to dismounted soldiers in the area; all with plenty of time for the soldiers to move away before the rounds ever landed. Based on this success, the Department of Defenses Joint Improvised Explosive Device (IED) Defeat Organization

(a) Insurgent launches mortar attack.

(b) Friendly radar detects the inbound round.

(c) Command systems display the threat.

(d) Soldiers in area notified before impact.

Figure 1.1. Ancile mortar strike warning scenario.

(JIEDDO) tasked the team with studying the suitability of the system to also warn soldiers whenever they approached the vicinity of a suspected IED.

As we started extending the architecture to encompass other threats such as IEDs, it quickly became obvious that there existed a large and important set of problems in both the public and private sectors for which Ancile could provide a viable solution [35]. Among these problems is the monitoring and control of emergency response personnel at the site of a man-made or natural disaster. This system could have, for instance, allowed medical personnel to quickly move to ad-hoc casualty collection points in the aftermath of hurricane Katrina. Additionally, the system could be used to warn key personnel when a tsunami has been detected so that they may, in turn, warn people in their vicinity. Fig. 1.2 shows our concept for Ancile.

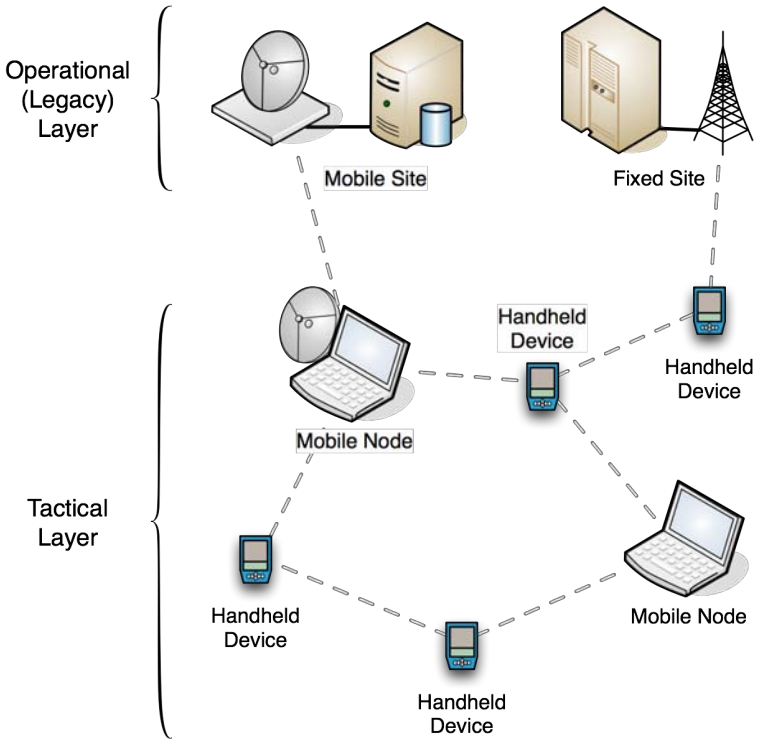


Figure 1.2. Ancile concept diagram

The Defense Advanced Research Projects Agency (DARPA) also took an interest in our work. The project manager for the Tactical Ground Reporting (TiGR) discussed

possible collaborations with us, but their need was too immediate to be compatible with our own schedule. They encouraged us, however, to continue developing Ancile with a view to eventually merging some or all of its capabilities with systems such as TiGR.

Besides the military and disaster scenarios where lives are at stake, Ancile lends itself to a myriad of other applications. Consumers could use the technology to receive warning messages when an item of interest is discovered at a particular price point in their vicinity. Drivers could exchange traffic information with others driving in the opposite direction so as to discover areas of congestion. Tourists could receive information of interest as they move through a foreign city. The possibilities are many.

1.2 Problem Statement

The problem we propose to solve is this: if information exists in the network, how can we ensure that all who need that information have it? If the nodes were static, then the problem would be a simple distribution issue that is easily solved using any of many well-known mechanisms designed for sensor networks [2]. Node mobility, however, significantly complicates the issue, since the information requirements of a node depend on its current location. Furthermore, node limitations such as battery power and local storage capacity further complicate the problem. We formalize this discussion in the next paragraphs.

Let C be the set of all categories of information that can exist in the model. C could then be thought of as the global catalog in a distributed database system. We then define R as the set of roles within the system. For instance, in Fig. 1.3, $C = \{crime, medical, traffic\}$ and $R = \{medic, police\}$. The relation $c : C \rightarrow R$ can tell us which roles are interested in a specific category of information. In the illustrated example, $c(medic) = \{medical\}$, and $c(police) = \{crime, traffic\}$.

Let $G = (V, E)$ be an undirected graph, where V is the set of network nodes and E is the set of connections between pairs of nodes $v_i, v_j \in V$. Every vertex $v_i \in V$ is represented

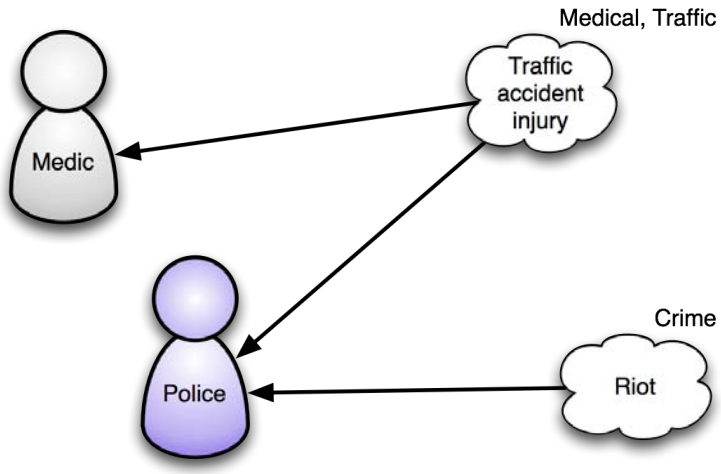


Figure 1.3. Roles and Categories of information.

by a tuple (l_i, r_i, k_i), where l_i is the n-tuple of the coordinates of the location of the node v_i in n-dimensional space. Note that n includes a time dimension, so that $n = 3$ actually represents 2-dimensional space with a time dimension. $r_i \subseteq R$ is the set of roles for (i.e. information of interest to) the node, and k_i is the set of information items that the node knows. Graph G is undirected since we assume all links to be wireless and bidirectional.

Let S be a set of information items where each $s_j \in S$ is represented by a tuple (g_j, u_j, p_j), where g_j is a tuple containing the time duration of the information expressed as a 2-tuple (t_0, t_1), followed by the n-tuples describing the n-dimensional region of space wherein this information is relevant. Table 1.1 illustrates some possible tuples g_j and their interpretation. The set $u_j \subseteq C$ is the subset of categories of this information within the set defined by the global catalog. The information item also has a relative priority level described by $p_j \in \{ROUTINE, PRIORITY, FLASH\}$.

Table 1.1. Sample information geometries in 2-dimensional space

g_j	Description
$((-\infty, \infty), (-\infty, \infty), (-\infty, \infty))$	Universal event (always, everywhere)
$((t_0, t_1), (-\infty, \infty), (-\infty, \infty))$	Everywhere event (limited time)
$((-\infty, \infty), (x_1, y_1))$	Always event, point in space
$((t_0, t_1), (x_1, y_1), (x_2, y_2))$	Limited time event, linear
$((-\infty, \infty), (x_1, y_1), (x_2, y_2), (x_3, y_3))$	Always event, polygon

Once an information item arrives, the node must decide whether to share it with other nodes, and then whether to keep it or discard it. We define a function $r : V \times S \rightarrow \{yes, no\}$, which determines if a node will relay a given information item. We also define a function $m : V \times S \rightarrow \{yes, no\}$, which determines if a node will maintain that information in its knowledge base.

Then the problem of pervasive situational awareness that we are addressing is finding r and m such that:

$$\forall v_i \in V, \forall s_j \in S, \quad l_i \subset g_j \wedge c(r_i) \subseteq u_j \implies s_j \in k_i$$

In other words, it must be true for all nodes and information that, if a node is inside the region of relevance for that information and the information is of interest to the node, then that information must be in the knowledge set for that node. This may seem like an obviously desirable condition, but the challenge is in finding local information sharing ($r(v, s)$) and information maintenance ($m(v, s)$) mechanisms that ensure, globally, that our implication holds. Furthermore, it is important to identify the conditions under which this model becomes unsustainable and fails.

1.3 Contributions

The main contributions of this thesis can be summarized as follows:

- Development of a conceptual framework for pervasive situational awareness. Situation awareness (SA) is the perception of environmental elements in space and time that are relevant to a particular individual or group. Our work has already laid the groundwork for an environment within which SA is disseminated to those who need it. This contribution was the subject of a recent article in a special edition of IEEE Internet Computing magazine [35].

- Development of a stochastic mechanism for transporting information across a mobile network. This mechanism is particularly well suited for dealing with adverse network conditions such as limited battery power, high node mobility which results in frequent topology changes, the sudden loss or destruction of nodes, and the presence of obstacles that impede direct communications. Part of this work was published in the 2009 International Conference on Information Technology: New Generations [37].
- Development of a mechanism by which nodes can maximize both local and global knowledge preservation and diffusion with limited power and storage capabilities. This allows the nodes to make room for new information by sensibly discarding knowledge only when it is likely that other nodes have chosen to preserve it, or when that piece of information is no longer relevant.
- An understanding of the conditions under which information is lost, exchanged, and preserved in a peer-to-peer mobile geospatial database system. This is an important contribution to the study of emergent knowledge processes (EKP), which are organizational activities that exhibit emergent processes of deliberation involving complex information and actors with unpredictable roles and prior knowledge [34]. Clearly, EKP is a topic of much relevance to our principal problem scope: emergency response and military scenarios.

1.4 Thesis Structure

This chapter has addressed the introduction of the thesis proposal; the rest of the document is organized as follows. We first develop the necessary background theory and provide an overview of relevant related work in Chapter 2. Chapter 3 introduces our stochastic geocasting model and presents the experiments we performed and the results we obtained in order to validate this model. This geocasting model addresses our need for a mechanism for transporting information to the regions wherein it is needed. We then turn our attention

to the issue of maintaining that information inside the region. In Chapter 4 we develop an analytical model for shared information within a region, and present the results of various analytical and empirical efforts to verify it. Armed with this theoretical foundation, we present a mechanism for maintaining a cooperative spatial-aware shared information cache in Chapter 5. Collectively, Chapters 4 and 5 address our need for a mechanism for maintaining information in the region and so, in Chapter 6, we discuss future work for this dissertation and our conclusions.

CHAPTER 2

Literature Review

2.1 Chapter Overview

It follows from our problem statement in section 1.2, that our work will build on the existing body of knowledge in the areas of mobile node communications and information management. This chapter summarizes this body of knowledge as it pertains to our specific problem. Section 2.2 will highlight the work by many others in the area of geocasting, which supports the first thrust of our own work in finding a function $r(v, s)$ that allows us to correctly move information around the network. The remaining sections collectively present the state of the art in the area of distributed, mobile, geospatial knowledge management, which speaks to our efforts at finding the knowledge management function $m(v, s)$ in our problem statement.

2.2 Geocasting

2.2.1 Overview

Geocasting is defined as geographic messaging [23, 38]. That is to say the delivery of messages to recipients based only on their presence within a given geographic region. It is analogous to multicasting but, unlike the later, delivery group membership is based solely on the recipient's location. Although the original work by Navas and Imielisnski [38] does not specifically

address node mobility, it is implicit in their paper that this is the likely direction of their ideas. It is later [27] that the application of Geocasting within mobile ad-hoc environments is studied in detail.

Fig. 2.1 illustrates the essence of geocasting. In it, a police officer has information about a bomb threat at the train station. Geocasting is the delivery of that information to all people at the train station. Note that intermediaries in this transmission need not be aware of the threat, but they are expected to assist in relaying the message. Furthermore, nobody in any region other than the train station receives the message, with the exception of those who relay it.

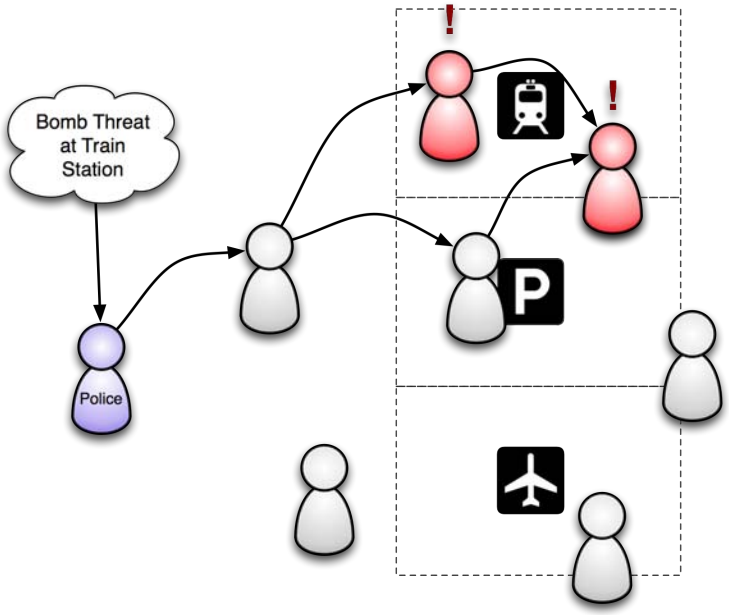


Figure 2.1. Geocasting a warning message.

At its most basic level, the geocast problem is solved by Cartesian Routing [39]. This solution relies on the assumption that each node (source, destination, and intermediary) knows its own geographic location and the locations of all its immediate neighbors. With this information, a node can determine whether it should relay a given message or not. If the node is further away from the destination than the source was, then it would probably not be

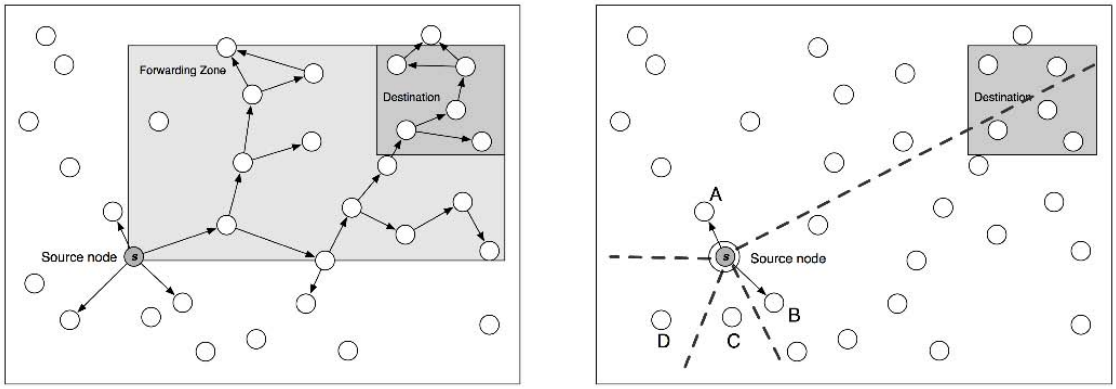
advantageous for it to retransmit it. An exception to this rule would be a situation in which no direct route exists between the sender and the receiver and the area of consideration for potential relay agents must be expanded until a viable route is found.

A variety of more sophisticated delivery mechanisms and protocols have been devised and there is ample documentation on all of them [32]. Generally, these protocols can be classified by whether they use flooding, limited (or regional) flooding, or non-flood routing to relay the messages to their destinations. Flooding is almost never used in real networks because of its tremendous bandwidth requirements. Among the most popular protocols that use limited flooding or non-flood routing are the Location Based Multicast [4] and Voronoi Region multicasting [51]. Though each of these protocols has a number of strengths, none of them are explicitly designed to address the issues of limited bandwidth and power common to pervasive mobile computing environments wherein embedded devices are the norm.

2.2.2 Location Based Multicast

Location Based Multicast (LBM) [4] comes in two flavors: a forwarding zone approach and a distance to target approach. In the first, a bounding box is defined by using the locations of the message originator and the geographical region of the message. All nodes within this region relay the message. Optionally a parameter γ can be used to expand (or contract) this region. In the second variety of LBM, each message contains the distance from the last sender to the center of the messages area of effect. A receiving node computes its own distance to the target area and compares it to the distance of the previous node. If the local node is closer to the message's area, then the node inserts its own distance into the message and relays it, otherwise the message is ignored.

LBM is simple to implement and produces good results in fairly regular topologies. Fig. 2.2 (a) shows a sending node s transmitting a message destined for the nodes in the zone on the upper right of the figure. All potential relaying nodes know the sender's location



(a) Location Based Multicasting. (b) Voronoi Region Multicasting.

Figure 2.2. Current Geocasting Approaches.

as it is included in the message header. Additionally, the destination zone is also known. Based on this information, nodes receiving the message can compute the forwarding zone depicted in light grey in the figure. All nodes within this region will relay the message.

2.2.3 Voronoi Region Multicasting

Voronoi Region multicasting [51] also uses the concept of forwarding zones, but differs in the way these zones are determined. Voronoi regions divide space into n regions around a given set of n points $P = \{p_1, ..., p_n\}$ so that any other point $p\prime$ is in region i if and only if it is closest to point p_i than to any other point in P. All nodes whose Voronoi regions intersect the geocast message's destination area receive the message via unicast or multicast transmission. In Fig. 2.2 (b), the sender s determines that nodes A and B have Voronoi regions that intersect the destination area. Node s unicasts the message to both A and B, but not to its other neighbors. Upon receiving the message, these forwarding nodes repeat the process until the message arrives at a node within the destination area. Within this area, the message is flooded by all nodes in it.

There are other geocasting protocols, but they rely on the election or selection of nodes that act as dedicated routers for a given region for a set period of time or until the topology changes significantly. Since the election of routing nodes in a peer-to-peer,

power-constrained environment is inherently unfair, we do not consider those approaches. The interested reader is referred to papers on the GeoGRID [31], and GeoTora [26] protocols.

2.3 Distributed Databases

2.3.1 Overview

A distributed database is one in which the data resides in multiple locations. Though this definition allows for data to live in multiple locations within the same computer, virtually all distributed databases involve multiple networked hosts. These hosts can run the same or different database management systems. They can also follow the same or different data schemas, though the later case requires a translation mechanism to ensure compatibility. An extensive coverage of these issues can be found in [40].

One kind of distributed database system is the federated database system (FDBS) [50]. This is a collection of databases that are autonomous and perhaps even heterogeneous that are integrated to some degree and behave cooperatively. The software that coordinates and controls the interactions among the databases is called a ferdarated database management system (FDBMS) as depicted in Fig. 2.3.

2.3.2 FDBS Categorization

We can categorize FDBS according to their distribution, heterogeneity, and degree of autonomy. The distribution dimension of this characterization refers to whether the data exists in disparate collections that share no commonality, or whether it is fully replicated and thus identical in every repository, or whether it is somewhere in between these ends of the spectrum. The benefits of distributed data in terms of response time, reliability and availability have been well studied for over 20 years [7].

FDBS can be heterogeneous with regard to the hardware and software on which they run, such as database systems with differing DBMS (e.g. PostgreSQL and MySQL), differing

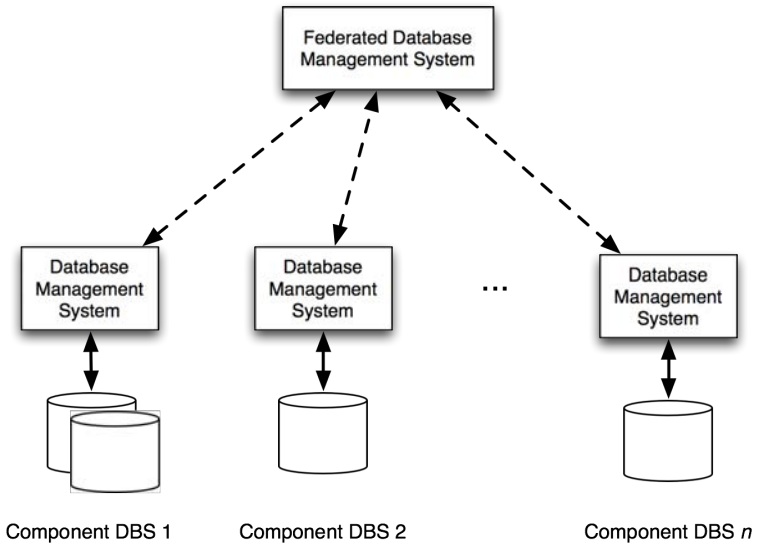


Figure 2.3. Structure of a Federated Database Management System.

networks (e.g. Ethernet and satellite), and differing query languages (e.g. QUEL and SQL). But FDBS can also be heterogeneous with regard to their semantics [22]. For instance the same information can be captured in different schemas using different table-column names, or conversely a given table-column reference may have different meaning in different systems. So, FDBS heterogeneity can be traced to the hardware or software on which the systems run, and to differences in the structure of the data.

Lastly, FDBS can be categorized according to the degree of autonomy of the component database systems. This autonomy can be classified into three types: design, communication, and execution. Design autonomy differences are a leading cause of FDBS heterogeneity. It refers to the autonomy that each component database has in choosing its own design with respect to any issue. Communication autonomy means that each component gets to decide with which others it is willing to communicate, and the conditions of these communications. Finally, execution autonomy speaks to the authority in choosing which queries or operations a component database will perform.

2.3.3 Distributed Transactions

A database transaction is the basic unit of work performed within a database management system. It can be defined as any single execution of a user program within a DBMS [44]. A distributed transaction is one which requires computational and/or information resources residing in multiple applications or hosts. Transactions must be atomic, consistent, isolated, and durable (ACID) [16], which is clearly a challenging proposition when one considers FDBS. Many approaches have been proposed for correct transaction processing in distributed databases [5, 24, 12].

The most widely-used model for enforcing the ACID properties in distributed transactions is the 2-phase commit protocol (2PC) [30]. The essence of this protocol is that transactions are performed in two phases: first, the transaction intentions are requested and acknowledged by all parties involved and, secondly, the actual operations (e.g. reads and writes) are performed on the stored data. The first phase ensures that all required resources are available and, if necessary, locked from access by other transactions. This phase ends when everything is ready and a commit command is issued to all parties. The second phase is the one that actually touches the data which, by now, is all available and ready.

2.3.4 Replication

Replication is an important safeguard against inevitable system failures [17]. The simplest form of replication is mirroring, in which identical copies of all data are kept at different sites. As the complexity of a distributed database increases, however, mirroring becomes prohibitive in terms of both cost and performance. All replication algorithms experience increasing costs and decreasing performance as a function of system complexity. This is particularly true of unstructured environments.

Peer-to-peer (P2P) networks are commonplace nowadays and they present some unique and difficult challenges to replication. A good approach to replication in P2P net-

works is to proactively replicate the data [10]. This approach can be uniform (i.e., all data get replicated at the same rate) or proportional (i.e., frequently-requested data is replicated more frequently than the rest). Since nodes with data stores are "blind" to what other nodes may have in their own data stores, this approach is based on a push of data from one node to its known neighbors. An optimal solution appears to be to balance the application of both uniform and proportional replication in P2P environments.

2.4 Spatial Databases

2.4.1 Overview

A spatial database is one that supports spatial data types in its data model and query language. Just as a conventional database supports whole and decimal numbers, a spatial database allows the use of points and polygons in space. Additionally, a spatial database should support spatial indexing and join methods [14]. Similarly to how we can select a value in a range in a database (e.g. age > 18 and age < 65), we can select a point within a polygon in a spatial one (e.g. all soldiers within area 52). We focus our discussion on spatial indexing and spatial join operations, since these are the critical components of a spatial database.

Most spatial databases use the concept of minimum bounding boxes to facilitate the processing of complex geometries. The idea is that a rectangle (or cube in 3-dimensional space) is significantly easier to compare to others than are complex geometries such as the ones that describe real-world objects. A minimum bounding box is the smallest rectangle that completely encloses a given geometry. Unless the underlying geometry is also a rectangle, the minimum bounding box will contain space that is not part of the geometry. For this reason, many operations on spatial data require two stages: one to identify candidate objects based on their minimum bounding boxes, and another to actually examine the geometries of interest from among the candidate list.

2.4.2 Spatial Indexing

Virtually all databases of practical size require the use of indexes to expedite access to records with specific indexed values. Without indexes, data access would be prohibitively slow in large data sets. Spatial indexing allows for quick retrieval of records with specific spatial values, such as "all restaurants within a mile of my location," by examining only records within the area of interest. While there are many mechanisms to accomplish spatial indexing, only two have found their way into the mainstream database systems: R-Trees and Hilbert curves.

R-Trees are dynamic index structures that allow indexing of objects in multi-dimensional spaces. More specifically, they are height-balanced trees whose leaf nodes contain pointers to data objects [15]. Non-leaf nodes contain entries consisting of an n-dimensional rectangle and a pointer to a child node containing all objects in that rectangular space. This creates a hierarchical structure of minimum bounding rectangles. Though we would normally use separate relations for moving objects (e.g. people) and static geometries (e.g. buildings), we depict both as part of the same relation in Fig. 2.4 for the sake of simple illustration of the R-Treee indexing concept.

Hilbert curves are space-filling curves first described by David Hilbert [48]. If we divide a region into a grid of subregions, a Hilbert curve will touch each subregion exactly once. Though these curves are not, strictly speaking, a spatial index, they do allow linear decomposition and ordering of multidimensional spaces. This process usually involves multiple passes, resulting in a hierarchy of curves of increasing granularity as shown in Fig. 2.5. Notice that the curve touches only once each region. The example we show only uses two passes, though in reality four or more are normally used. The method for numbering the resulting grids varies and we show the traditional Hilbert implementation.

Once this decomposition is completed, the spatial records can be indexed using a linear index such as the ubiquitous B-Trees. There are many ways of assigning spatial objects

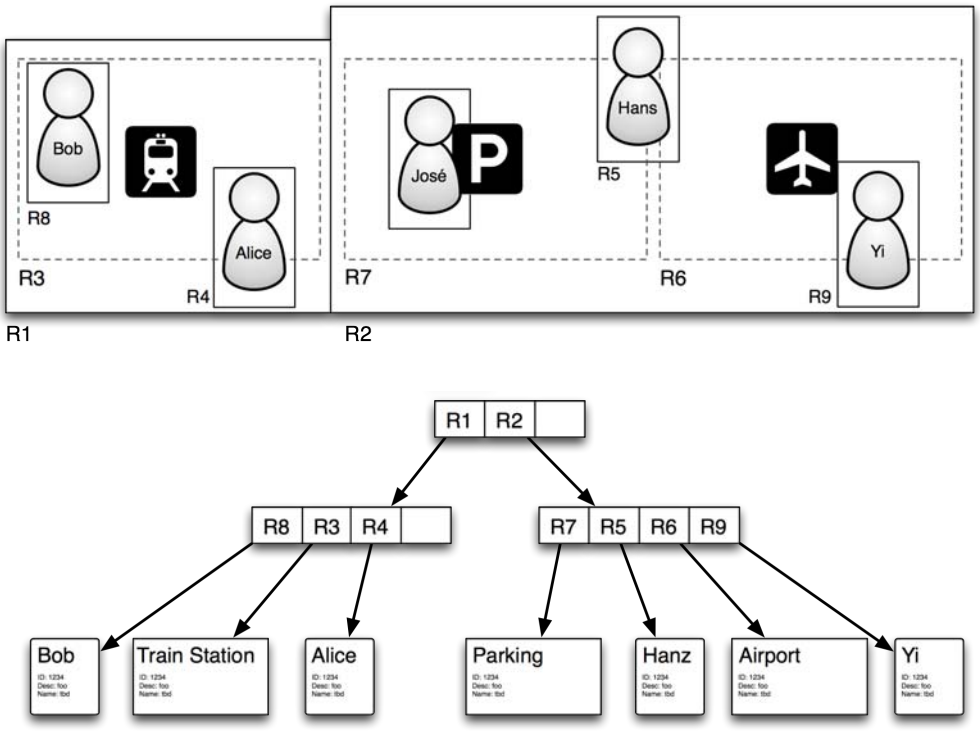


Figure 2.4. Sample R-Tree in 2-dimensional space.

to grids. Normally, we want to know the grids in which any part of an object is. After the two-pass decomposition in Fig. 2.5, we can say that Bob is in grids 1, 2, 3, 4, 5, and 6 and Alice is in grid 9, 10, 11, and 12. The example shows an interesting feature of space-filling curves in general and Hilbert curves in particular: that they tend to preserve fairly well multi-dimensional spatial proximity in the resulting linearization. Once the space has been decomposed can build an index using B-Trees where the locations of our objects are in linear space.

2.4.3 Spatial Join Operations

A join operation is one that combines records from two or more relations (or tables) that match specified criteria. Joins can be seen as a subset of the cross product of two relations. An example of a simple join in a traditional database with two relations *Employee* and *Department* would be to select all employees in a given department. In the case of a spatial

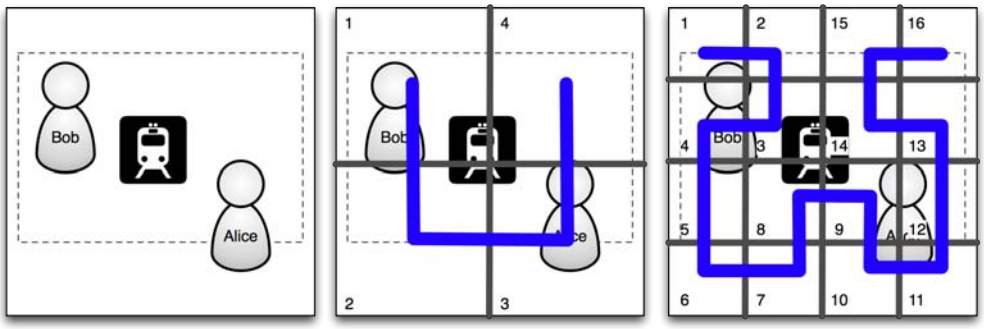


Figure 2.5. Hierarchical decomposition of 2-dimensional space using Hilbert curves.

database an example join would be to list all restaurants within a given city. Though the results are conceptually the same in both the traditional and spatial cases, the way in which the join operations are performed is quite different. In particular, spatial join operations normally require two steps: first, a set of candidate objects is chosen based on their minimum bounding boxes, and secondly, the candidates are examined in detail to determine if their actual geometries match the join criteria. This second pass has a significantly higher computational cost, so the overall system performance hinges on a first pass that correctly filters out most of the areas that should not be carefully examined.

2.4.4 Nearest Neighbor Queries

A type of query we would expect to encounter in a spatial database is to find the k nearest neighbors to a given point in space. For instance, we may want to know the four closest police officers to the scene of a looting incident. While it is possible to execute these queries as containment queries on an R-Tree, it is a very inefficient approach.

A better solution is to use a specialized algorithm such as the one proposed by Roussopoulos, Kelley, and Vincent [47]. This approach is built on R-Tree indexes. They propose a branch-and-bound depth-first search along the R-Tree wherein, at each node of the tree, all entries are examined and ranked in terms of the minimum distance between their minimum bounding rectangles (MBR) and the point of interest. The search then selects the child node

with the least distance. The search continues until a leaf node is reached, whereupon the node is searched for an object matching the search criteria. At this point, we have the first of our nearest neighbors. To find the next, we simply look at the ordered list of nodes that was built during the depth-first search and continue this process until all k nearest neighbors have been found.

2.5 Mobile Databases

2.5.1 Overview

A mobile database is one in which some or all of its nodes (clients, servers, or both) are mobile. This situation implies the use of wireless networks to connect a subset of the nodes. Clearly, mobility, introduces many new challenges, of which two of the most interesting are the existence of location dependent data and the distribution of this data [29].

Location dependent data (LDD) is that which is tightly coupled with a particular region in space. For example, when we refer to the airport, we may be talking about the Luis Munoz Marín or the John F. Kennedy international airports depending on whether we are in San Juan, Puerto Rico or New York City (respectively). The response to the query "how far am I from the airport" depends on my location when I issue it. This means that, when dealing with LDD, the system must know the location from which the query was issued.

A related issue is how to distribute this LDD throughout the system. While this would seem like a straightforward issue in most instances, it becomes complex when we deal with distributed transactions and replication. While different approaches exist [43] to address the issue of transaction processes in mobile databases, we have not yet arrived at a universally accepted model.

2.5.2 Transaction Processing

Approaches to mobile transaction processing can be separated into two groups: those that assume the transaction will occur at a fixed host in its entirety, and those that assume that at least some part of the transaction will take place at one or more mobile hosts. Since the second approach is clearly relevant to our efforts, we will concentrate our attention on it. Within it, we are specifically interested in the case where transactions exhibit distributed execution among peers. Unfortunately, none of the existing approaches to mobile transaction management adequately support this case [49].

2.5.3 Replication

Three of the most important issues to be considered when addressing the problem of mobile database replication are node power consumption, real-time application requirements, and network partitioning [41]. The issue of power stems from the fact that mobile nodes almost always are powered by batteries, which have a very limited lifespan. Real-time application requirements are similarly important, because the replication approach that is used cannot interfere with these requirements, which are oftentimes life-or-death. Lastly, the issue of network partitioning is clear to anybody who has ever had a cell phone call dropped; service interruptions and disconnects are a fact of life in wireless networks. None of the existing data replication techniques address all three issues and, thus, this continues to be a very active area of research.

2.6 Data Caching

2.6.1 Overview

Users of computer information systems are likely to access data in patterns that exhibit temporal or spatial locality [3]. This means that the data will be close to each other in terms

of time (i.e., data sets accessed at approximately the same time) or space (i.e., data sets comprised of items that are close to each other). Additionally, users in the same locality tend to access similar data. The exploitation of these trends has resulted in significant performance enhancements through the use of data caches. A data cache is a temporary store of information that is close to the user and will likely be requested in the near future.

Caching can occur at many levels within an information system. Typically, however, we speak of either caching servers and caching clients. A caching server is one that is electrically closer to the requesters than the data store. Many organizations, for instance, deploy web caching servers whose purpose it is to cache the web pages accessed by the users so that subsequent request for those pages can be satisfied locally, without having to wait for communications to and from the distant web server. Clients can similarly cache data for different processes, sessions, or even transactions occurring at one client device.

2.6.2 Query Result Caching

Even with R-tree indexes, spatial queries are expensive in terms of both processing at the server and, more importantly in our current line of work, in terms of transmission costs of moving the data from the server to the requesting node. Obviously, it would be beneficial to reuse the results of past queries to satisfy future ones. In order to do this, there exist a variety of both local (i.e., intra-nodal) and cooperative (i.e., inter-nodal) caching schemes. At the most basic level, a node can store the results of its queries in hopes that they may be able to satisfy future queries from the application layer. This simple caching mechanism fails to exploit items that may be stored in neighboring nodes. In terms of cooperative caching, two promising approaches which have been developed by others are the 7DS architecture [42], which shares cached data among directly connected neighbors, and a more flexible approach proposed by Liangshong Yin and Guohong Cao [55], which allows nodes to exploit the caches of others that are not directly connected.

2.6.3 Cooperative Caching

It is possible federate client systems so that they share the contents of their individual caches [11]. Collectively, they are thus able to cache more data than any of them individually could. Central to the ability to perform effective cooperative caching is the existence of a local area network that is high speed when compared to the wide area network used to connect to the server. Additionally, there is a presumption that the load on the processors is negligible or at least very small.

2.6.4 Caching in Mobile Ad-hoc Networks

Huang, et al. [21] discuss a comprehensive approach to collaborative spatial data sharing among lightweight devices that exploits local caches as well as those of neighboring nodes. Their work is based on the notion that all (or at least many) mobile devices are connected to data servers over narrowband links, and to each other over broadband, peer-to-peer (P2P) connections. Based on this premise, they address the issue of reducing the cost and response time of spatial queries. Nodes within their framework cache data and keep track of nearby nodes that contain other data of interest (and which doesnt fit in their own caches) by using routing tables. As in the case of broadcast disks, this approach is optimized for read operations, but updates and inserts are supported by having the centralized server broadcast the existence of the new or updated data items. It is this dependence on central servers that makes this approach somewhat less than ideal to support our own efforts, in which servers may or may not be a part of the network.

Yin and Cao [55] describe a cooperative caching scheme for nodes in mobile ad-hoc networks (MANETs) in which both query originators and query forwarders perform caching. Their scheme is cooperative, because of the caching performed by the relaying nodes. In their paper, they show that the best results are obtained when nodes are allowed to cache both data items and also the paths to nearby nodes that have those items. This means that, once a data item has to be removed from the cache, a node may still remember the nearest source

for that data. This facet of their scheme is of much interest to us, because it can be used as the basis for a more elaborate collaboration scheme.

Just as Yin and Cao expanded the contents of a traditional cache in order to support routing information, Hu, et al. [20], similarly broaden the concept, but in order to support semantic caching. In semantic caching, nodes cache not only the data, but also the semantics of the query which yielded it. In order to efficiently support semantic caching in spatial databases, Hu and his team store the indexing structure which supports the query. The inclusion of a partial index allows nodes to determine which future queries could also be supported by the cached data, which makes their approach proactive. As promising as this is, it does not incorporate node cooperation, though we believe its key elements are amenable to our efforts.

Huang, et al. [21], proposed a fairly straightforward mechanism for nodes to keep track of the data which their neighbors are caching. At the heart of their approach is a string that is as long as the number of data items in the database and which nodes use to advertise which items are in their caches. Obviously, this assumes that all nodes are aware of the identity (though not necessarily the attributes) of each data item in existence. This information is periodically provided to all nodes by the server to maintain system-wide coherence. Though this use of cache signatures is very promising for our own purposes, their dependence on nearly constant connectivity to the server violates one of our basic constraints.

Finally, Chow, et al. [8], proposed GroCoca, an approach to mobile peer-to-peer (P2P) cooperative caching that exploits the tendency of mobile nodes in certain situations to cluster into groups based on either mobility, or data needs, or both. These tightly-coupled groups (TCGs) create opportunities for significant collaboration among group members. Though we can certainly exploit their use of TCGs in our own work, we have to relax group membership rules in order to support our specific needs. Moreover, Chow and his colleagues make an implicit assumption, as did Huang, et al., that nodes know the extent of the items in the global database and use this to create their own cache signatures. Again, we cannot

assume this, so we had to find a workaround.

CHAPTER 3

A Stochastic Approach to Geocasting

3.1 Overview

In this chapter we discuss how we propose to reliably and efficiently move information across our network. Recall that this helps us address the first part of our problem statement from section 1.2, which calls for us finding a function $r(v, s)$ that determines whether a given node will share information with others or not.

A high-level view of our proposed system is depicted in Fig. 3.1. Ancile exists in what is labeled as a tactical layer, but interfaces with legacy database systems in the operational layer using a variety of communications means including, but not limited to satellite, and line-of-sight radio. At our level of abstraction, the underlying communications technologies are unimportant, provided that they allow for broadcast communications among neighboring nodes. It is at the node that all that makes our system unique takes place.

We assume that nodes possess only a limited wireless communications capability. The radius of communication for any given node is significantly smaller than the total area in which the nodes exist. Whenever two nodes come within radio range of each other, they automatically establish a connection over which messages can be transferred. This connection persists until such time as the nodes move away from each other and the distance between them exceeds their radio range.

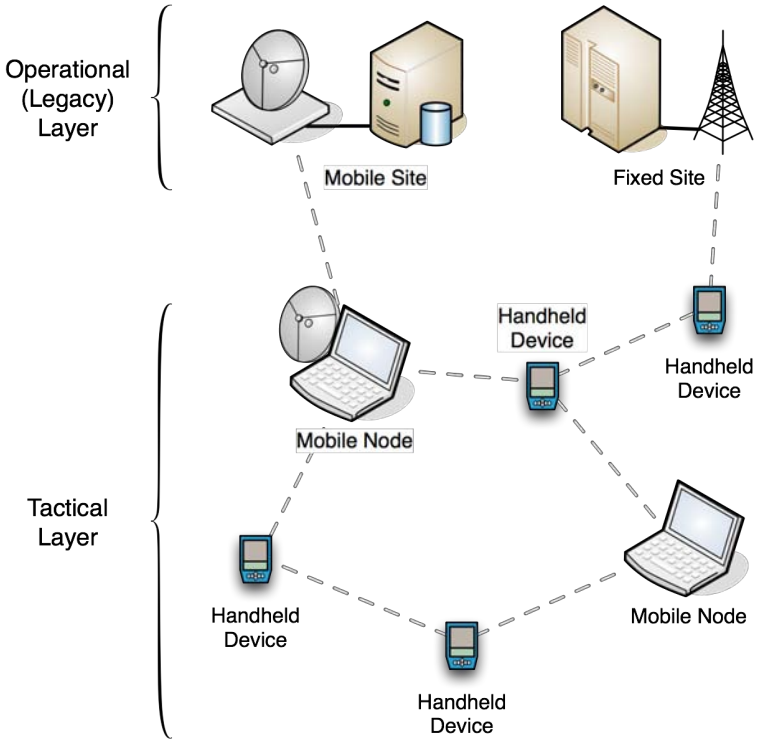


Figure 3.1. High level Ancile architecture

Nodes also have a limited power supply. A small but significant portion of this power is consumed every time a node transmits a message. If the power is expended, then the node is unable to participate in the network any longer. We assume that the power consumed by a node in receiving messages is negligible when compared with the power required to transmit.

Part of our interest is in developing a system within which power consumption is fairly spread among all nodes so that we dont see overly active nodes that soon run out of power. Here, we search for a function $r(v, s)$ that allows us to correctly move information around the network as specified in our problem statement in section 1.2. Recall that the problem we are solving is to find that relaying function together with a management function that, together, ensure that all nodes in the network have all the information that pertains to them.

Messages vary in the nature of addressing. A message may be exchanged between specific nodes (e.g. when a node is responding to a query from another) or from a node to a region (e.g. normal geocasting). Though different types of messages are handled differently

by the nodes, they all share some basic metadata that is important for the functioning of our system. This information is structured as shown on figure 3.2, and described below.

- **Header length.** Since the geometry descriptor is of variable length (depending on the number of vertices in the geometry), the header length is used to determine how many points are specified.
- **Type.** We anticipate the need for different message types, so we include this data item here. Examples of message types are queries, configuration changes, and security key exchanges.
- **Flags.** Our work to date only uses the HELP flag described in section 3.5 below. Still we anticipate the eventual need for other flags.
- **Message ID.** This is required to ensure that duplicates are discarded.
- **Source.** Both the identity and the location of the sender are recorded. The former so that a query response can be sent to the right originator and the later allows intermediate nodes to determine if they are in a location conducive to efficiently relaying the message.
- **Destination.** As in the case of the message source, we include both the identity of the destination node if it is known (e.g. when responding to a query), and/or its location. These values are used almost exclusively in query responses.
- **Last.** The location of the last hops transmitter is useful in determining whether a message is getting closer or farther from its intended destination.
- **Geometry.** The area of effect of the message, whose length varies depending on the number of vertices specified in the geometry's enclosing polygon.

3.2 Utility Functions

Our approach to addressing the issues of limited power and bandwidth within the geocast problem space is to use utility functions to decide when to transmit a message. A utility

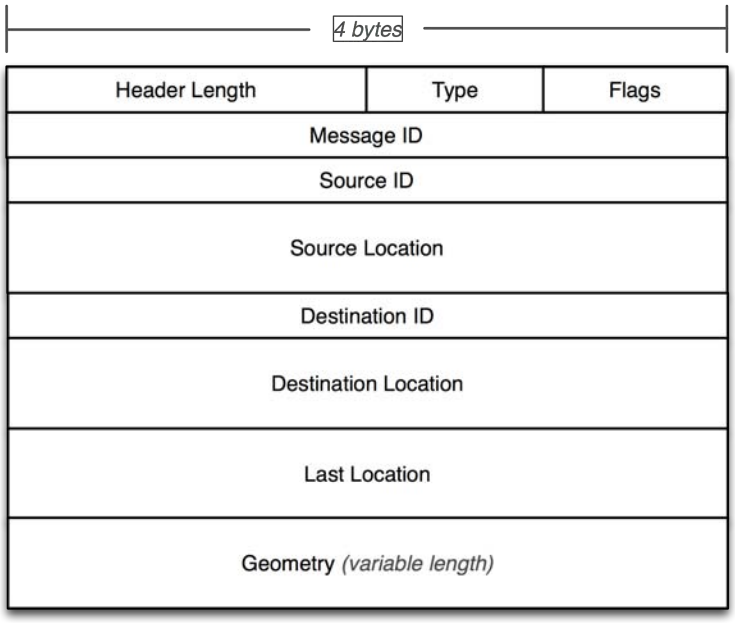


Figure 3.2. Message header structure

function maps a set of items to a value which represents their utility. Though the term originates in economic theory, it has become an important part of the field of game theory. Essentially, a utility function allows a node in a mobile ad-hoc network to determine how attractive a given course of action is with regards to a specific message.

When a node receives a message, it may take one of three externally visible actions: respond to it, relay it, or ignore it. Nodes can respond to messages when they are queries for information that the node possesses. Normally, a node which is able to respond authoritatively to a query is required to do so and, therefore, need not compute the utility of its choices. This trivializes the case in which a node can respond to a query. We will not address this issue further.

Nodes can also relay a message so that it reaches other nodes. When they choose to relay, nodes want to maximize the amount of information they acquire per unit of energy expended. All transmissions incur a cost in battery power. A node that ignores a message incurs no cost, since it does not transmit. Its probability of obtaining any utility, however, may be lower since it is not helping the information flow through the network and must rely on others to do so.

This means that, when a node relays, it does so in the hopes of obtaining some useful information, either from the response of some other node to that message or by motivating other nodes to relay some other messages that may be in turn useful to the local node. Quite simply, nodes tend to behave selfishly by directly or indirectly trading energy for information.

3.3 Utility of Relaying

Our utility function estimates the relative utility of relaying a message. This utility is an indicator of how likely the retransmission is to eventually cause the node to acquire useful information. The function parameters capture information about the message, the neighborhood of the node, and the state of the node itself. Clearly, there are many possible ways of capturing information about these issues. In our work, we chose straight-forward parameters in an effort to keep the computations as simple as possible. The main parameters we chose for computing the utility of relaying a message can be summed up as follows:

- **Proximity (p)**: If the node is not part of a fairly direct path between the sender and the destination, then it probably should not relay the message. We measure this as the distance between the local node and an imaginary line connecting the messages source and its destination as shown in Fig. 3.3. The formula is a simple manipulation of the Pythagorean Theorem using the dot product of the two known vectors $\vec{si}$ and $\vec{sd}$.
- **Degree (d)**: The degree of a vertex in a graph is the number of edges connecting it with other vertices. If a node in our framework has a lot of connected neighbors, then it may not receive much utility from relaying a message since chances are that at least one of the neighbors will do so.
- **Battery Power (b)**: If the node has consumed a lot of its battery power, then it probably should not relay messages unless it is clearly best to do so. Conversely, nodes will be less discriminating when relaying messages if their batteries are fully charged.

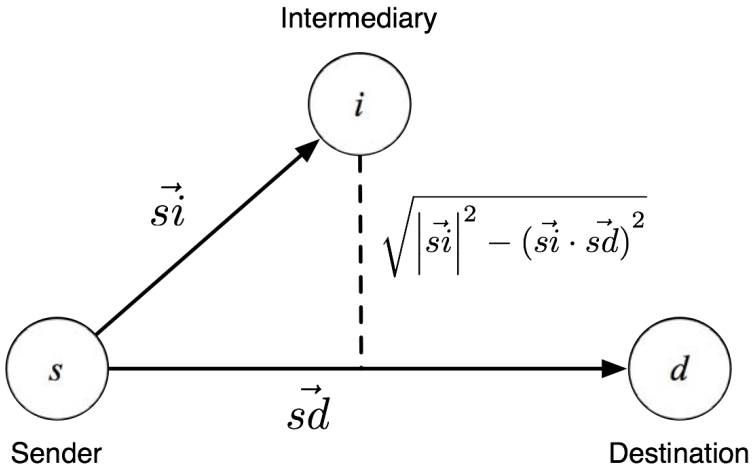


Figure 3.3. Proximity to most direct message route

Conceptually, we expected our relative utility function to have a maximum value of one and to asymptotically decay to zero as the message became less unattractive to the node. This asymptotic behavior means that, for all practical purposes, there is always a chance, small though it may be, that the node will relay a message regardless of how useless it may seem. We also thought this decay in utility should be non-linear, since we want the nodes to be fairly picky about the information they transmit. For these reasons, we chose to implement an inverse exponential function as shown in equation 3.1, which calculates the utility to node i of relaying message m at time t.

$$u_{i,t}(m) = e^{-\frac{p_{i,m} d_i}{\alpha b_{i,t}}} \tag{3.1}$$

The only term that has not been discussed hereto is α. This is a scaling factor for the other parameters. It ensures that magnitude of the numerator and denominator of the exponent are similar. Were this not the case, the numerator would tend to be at least an order of magnitude greater then the denominator, which means that most nodes would not relay messages. Obviously, as the value of α grows, the number of nodes retransmitting will be greater, which could lead to wasteful transmissions unless checked.

We ran a separate set of experiments using a variety of random network topologies to study the effect of α on the delivery rates and costs for otherwise identical scenarios. As can be seen from Fig. 3.4, α can assume any value that is at least 50 regardless of topology. However, increasing this value will result in additional costs with little or no improvement in the delivery rates. We, therefore, use an α value of 50 in the remainder of this work.

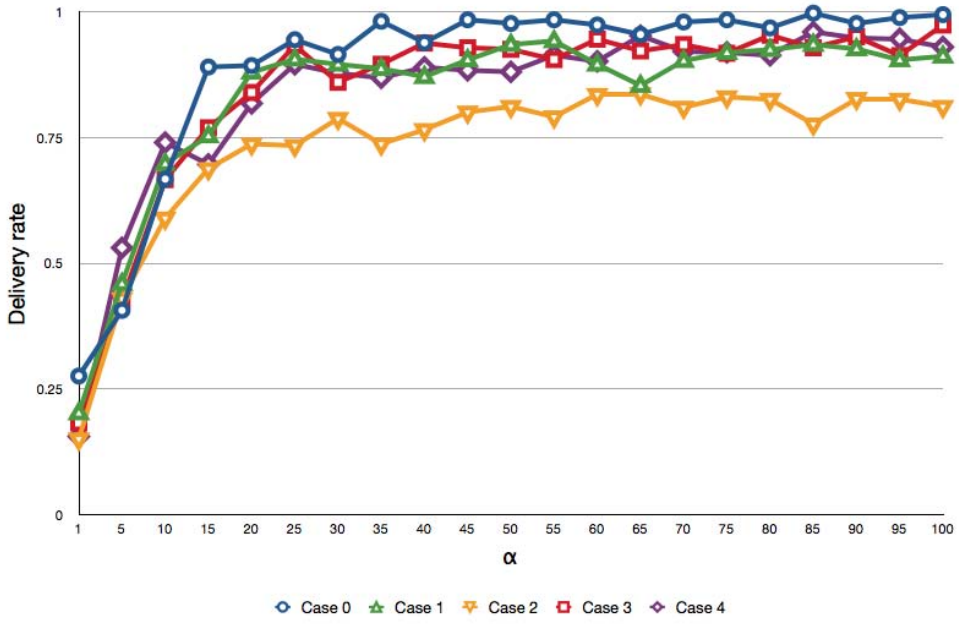


Figure 3.4. Effects of α on delivery rates.

3.4 Stochastic Behavior

Once a node computes the relative utility of transmitting a message, it must then determine whether or not it will do so. Our approach is to use a simple pseudo-random number generator to produce a uniformly-distributed value between zero and one. This value is compared to the computed utility and, if it is less than or equal to it, then the message is transmitted. This means that, while the messages will usually follow the best routes (from the perspective of the individual nodes), they will also sometimes follow some very unlikely

ones. We believe the non-deterministic nature of our approach makes it more effective at delivering messages under adverse network conditions. This is the characteristic of our work that distinguishes it from all others.

3.5 Obstacle Avoidance

We observed that our approach, just like every other existing algorithm we studied, suffers significant performance degradations as the nodes in the network cluster into strongly connected sub-graphs which are, in turn, sparsely connected to each other. In other words, as the network becomes less regular, regions devoid of nodes start to form and grow and, once the diameter of them exceeds the effective communication range of the nodes, these empty regions become obstacles to effective geocasting.

Fig. 3.5 shows two graphs, the first with a fairly regular node layout, and the second showing the same nodes forming strongly connected components. This clustering effect is very common in social networks, which describe a broad group of applications for MANETs. Our interest is primarily in emergency response and military scenarios. Within these, we can expect a very strong degree of clustering, not only because of the inherently hierarchical structure of these networks, but also because of their mission-focus. By mission-focus we mean that groups of people are required to work together within a limited geospatial region for the purpose of accomplishing a specific mission or task.

All existing approaches to the geocasting problem, including our own utility-based one, fail to perform adequately in networks which show this clustering effect. Though we defer a complete discussion of this issue until the next chapter, table 3.3 illustrates this phenomenon in a fairly extreme case of node clustering. In order to address our objective of pervasive information sharing within emergency response and military MANETs, we need a transmission mechanism that performs acceptably regardless of the regularity or clustering in the network of interest.

Many messages are not delivered because there are one or more obstacles between the

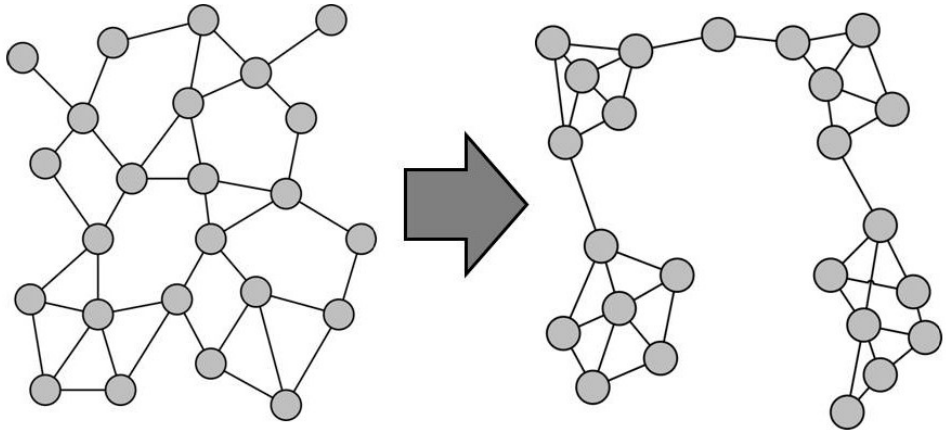


Figure 3.5. Clustering of nodes over time

messages authors and the destination areas. We define an obstacle as any region of the graph devoid of nodes and whose diameter exceeds the transmission range of a node. As a result of this, if a node that is adjacent to a dead zone decides to relay the message, the message will be lost because no other node will relay it further. Fig. 3.6 illustrates an obstacle.

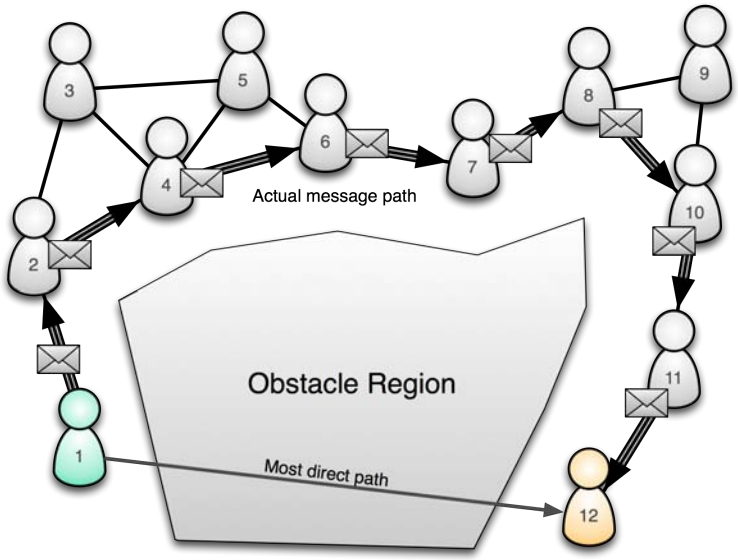


Figure 3.6. Message delivery around an obstacle

In the scenario depicted in the illustration, the successful message delivery hinges on whether or not node 2 relays it. However, this would normally be very unlikely since node 2

is fairly removed from the line connecting the sender (node 1) and the destination (node 12). More often than not, we would expect the message to fail to reach its destination. What we would like to see is the message hugging the obstacles boundary as it works its way around it. Once the obstacle has been cleared, then normal relaying rules could once again take effect.

To address the problem of obstacles, our system needs to be augmented by a mechanism allowing nodes to determine their adjacency to an obstacle to the transmission of a given message. To do this, we implemented a HELP flag in the messages [37]. Prior to sending any message, the sending node will check the direction to the destination zone and see if any nodes are within 45 degrees of the azimuth to the center of this target region. If at least one neighbor is in this arc, then the message is transmitted normally, since there are no apparent obstacles in the way. Otherwise the node sets the HELP flag on the message to signal receiving nodes that this message will probably need a circumspect route to its destination.

Upon receiving a message for possible relay, nodes perform their utility calculation as normal, but they then examine the HELP flag on the received message. If that flag was set, then the utility of relaying it is incremented by a factor β. The resulting utility is capped at a maximum value u_{max} to ensure we preserve the stochastic behavior of the nodes. In our work, we used $\beta = 2.0$ and $u_{max} = 1.0$ with good results.

However, if a node determines that it needs to set the HELP flag on a message and that message was received with that flag already set, then the node will maximize the utility of sending the message with regard to distance parameters. This means that, unless some other parameter dominates the equation (e.g. low battery power on the node), then the nodes probability of relaying approaches 1.0. If, on the other hand, the HELP flag was not already set on the message, but the node will set it, then it will ensure that the probability of relaying it is at least 0.5.

To sum it up, nodes that detect a possible obstacle between themselves and a message

set the HELP flag on that message prior to transmitting it. When a node receives a message with a HELP flag set, the utility of relaying that message will be at least 0.5, which gives the message at least even odds of being relayed. It is important to note that this affects nodes that normally would have a very low likelihood of relaying these messages.

3.6 Simulation Implementation

The utility functions and HELP flag mechanisms described above are implemented as methods in a Java class that models the individual nodes. In order to support a stochastic, discrete event simulation of the behavior of aggregate nodes, we built our models on top of a Java simulation framework developed by Ross McNab and Fred Howell called SimJava [19]. This simulation framework offers many conveniences, including the ability of graphically displaying a simulation within a Java applet.

The main class is the AncileSim class, which extends the Anim_applet of SimJava. Its main purpose is to read the configuration files for both nodes and message traffic, create the appropriate objects, and schedule the message transmissions. Additionally, this class computes the shortest paths for all messages from their author to all nodes affected by them. After the simulation run is complete, the AncileSim class takes care of collecting all statistics of interest. AncileSim also renders the graphical depiction of the simulation as shown in Fig. 3.7.

The main class that facilitates the measurements we needed is the the MessageTracker class. It is essentially a data class, but it implements the Java synchronized interface to allow each thread in the simulation to safely store its values in the various attributes. For each message, this class keeps track of the nodes who should receive it, the shortest path for delivery, and the actual number of transmissions (including relays) of the message. A reference to a MessageTracker object is passed to each entity in the simulation.

The main entity is, of course, the Node. This java class implements all the necessary

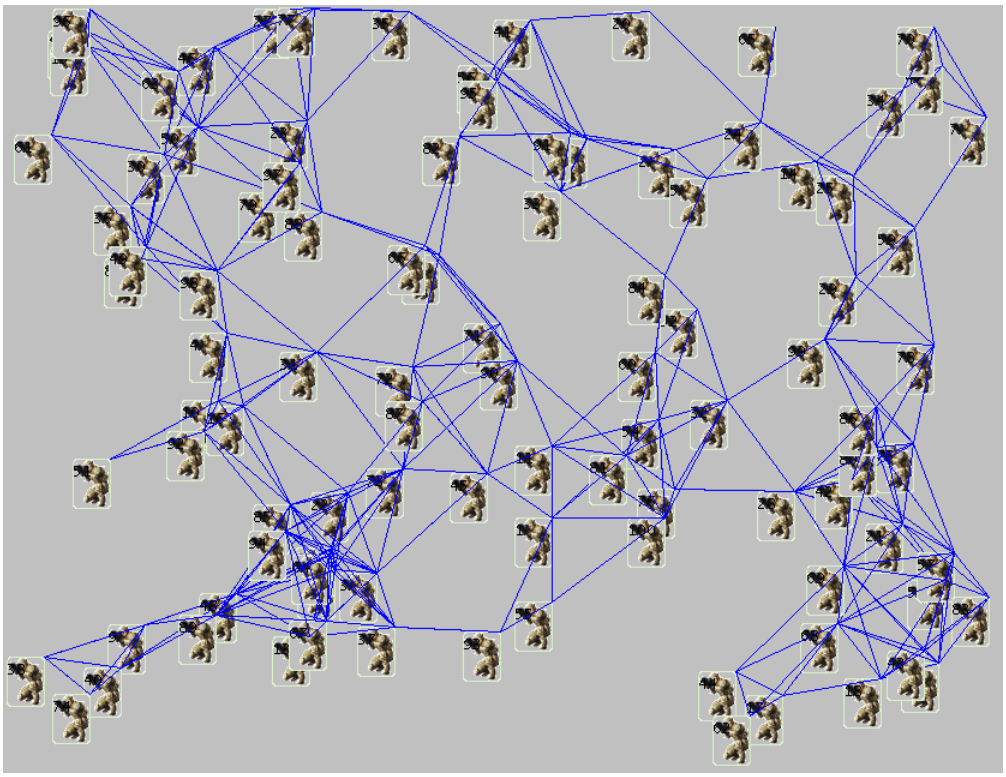


Figure 3.7. SimJava applet animation screen shot

logic for the creation, relaying, and consumption of messages in the network. Each node can be set to behave according to one of five strategies: flooding, utility relaying, forwarding-zone LBM, and distance-based LBM, and Voronoi region forwarding. The Node class maintains the current battery level for the node and disables it if it is completely spent. Each Node instance runs as a separate thread. Additionally, each node can have the HELP flag feature described in section 3.5 activated or deactivated independently of the forwarding strategy selected.

The remaining two classes are the Message class and the Location class. Both are data repositories. The Message class provides the ability to clone an instance so that nodes can generate their own new Message instance when relaying. This is needed because in Java objects are passed by reference and a node updating the sender value would overwrite the value set by another node in another thread for its own relay of that message.

The node and message data were stored in files which were read by the simulation

environment prior to each run. These configuration files allowed us to easily reproduce each experiment. The nod file contains the node identifier and x and y coordinates for each node. The message file contained the identifier of the sending node, the x and y coordinates of the center of mass of the area affected by the message, and the delay in simulation time units from the start of the simulation to when the message would be originally transmitted. All messages were given a fixed area of effect of 100 units by 100 units centered on the coordinates specified for that message in the message file.

We conducted five runs of each scenario for the stochastic models (with and without the HELP flag) and aggregated the results. For the deterministic models, we only conducted one run for each of the four topologies. For each message, we recorded the distance between its source transmitter and its point of interest. Prior to running the simulation, we determined which nodes should receive each message and calculated the best case cost of transmitting each message using a breadth-first shortest path algorithm. During the simulation, we kept track of the cost incurred for the transmission of each message, as well as the number of nodes within the area of interest which actually received the message. These results were dumped by the simulation after each run whereupon we analyzed the results.

3.7 Performance Study

At a conceptual level, we were confident that we had found an effective and efficient mechanism to move information around the network. Recall from our problem statement in section 1.2, that part of our problem was finding a function $r(v, s)$ that could accomplish this. This chapter describes the experiments we conducted to validate our work in that regard.

3.7.1 Experiment Design

We wanted to obtain a relative benchmark for the performance of our model in a realistic environment. Specifically, we wanted to see how it compared to the worst and best case

scenarios in terms of aggregate cost. This cost was defined in terms of total number of transmissions per distance traveled by the messages. The worst case cost is obtained when flooding is used because in this situation every node is required to transmit every message exactly once, which results in a constant cost of nm, where n is the number of nodes in the network and m is the number of messages originated. The best case scenario, on the other hand, was defined as the shortest path between the messages author and all nodes in the affected area. To compute this shortest path, we used a simple breadth-first search algorithm that looked for all nodes in the area of interest and computed the longest distance to reach all of them. Clearly, the best case scenario is not realistic in any viable MANET, but it is still useful in terms of understanding the performance of our own approach.

The network used in our experiments consisted of approximately 100 nodes spread out over a rectangular area of 4,800 square units. The exact number of nodes depends on the scenario under study. These scenarios are discussed in later in this section. Each node had a communications range of 100 units and automatically established bidirectional communications links to all neighbors within range. We used four types of network topologies detailed below.

1. In the first topology, depicted in the top left corner of Fig. 3.8 all nodes were placed regularly on a lattice configuration where the horizontal and vertical distance among the nodes was constant. This topology was used as the baseline case; all protocols should perform reasonably well, since there are no obstacles or adverse conditions.

2. The second topology was identical to the first, but an exclusion region of 900 square units was placed in the middle of the simulation area as shown in the top right of Fig. 3.8. All nodes in this region were removed from the simulation. The intent of this topology was to observe the difference in performance in the pressence of a significant, but not severe, obstacle.

3. In order to test the performance of the mechanisms under pathological circumstances,

we further modified the second topology by expanding the node exclusion zone (i.e. obstacle) to twice its previous size and ensuring it went all the way to one edge of the simulation area. The bottom left of Fig. 3.8 depicts the placement of nodes for this scenario.

4. Lastly, we ran a scenario were the nodes were randomly placed using a uniform distribution. We show the placement of nodes for this scenario in the bottom right of Fig. 3.8. Note, however, that we ensured that the resulting topology represented a connected graph, since message delivery would be impossible otherwise.

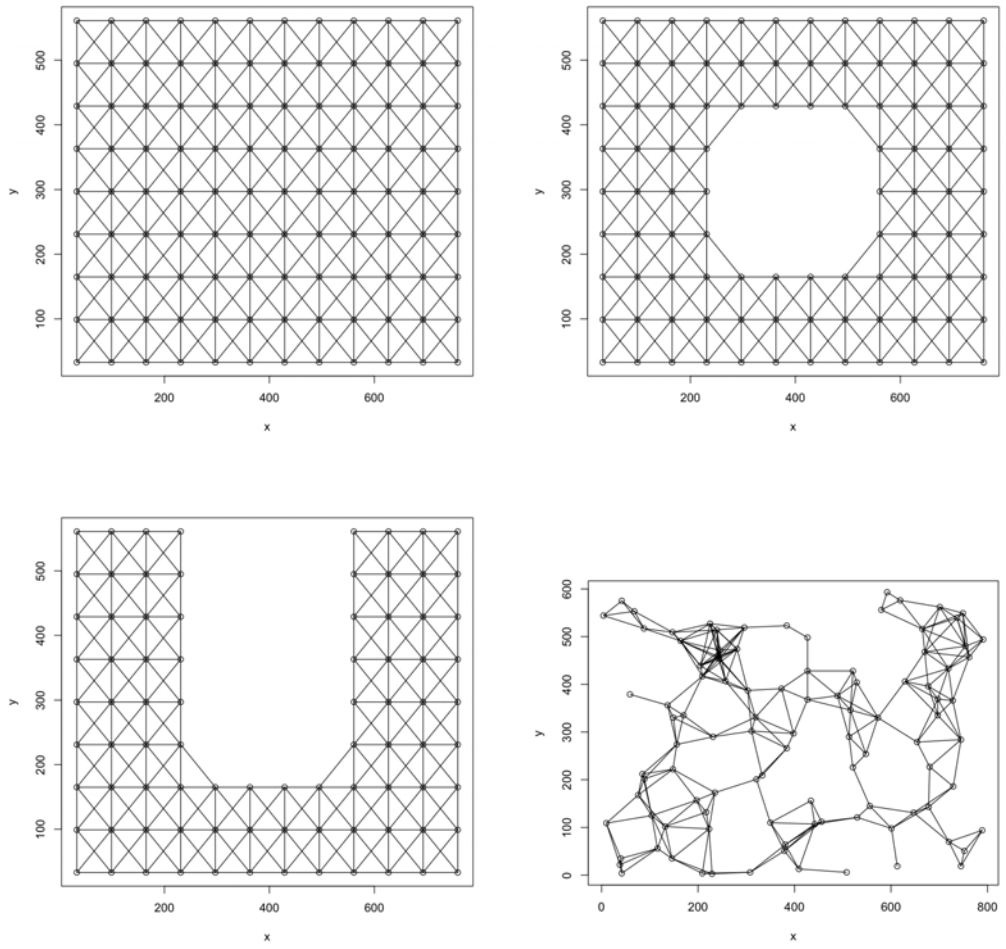


Figure 3.8. Node layouts

We ran each simulation using fixed message traffic which was specified in a configuration file as described in section 3.6. Scenarios 1 and 4 use identical message source-destination pairs, which were randomly generated using a uniform distribution of source nodes and destination coordinates. Scenario 2 uses a similarly generated file, but no source nodes or destination coordinates are permitted within the central exclusion zone. Finally, in the pathological scenario (number 3), we only permitted nodes with y values greater than 400 to originate messages. This forced all message traffic to travel at least a third of the height of the graph. In other words, the mechanisms being tested would have to find a very circumspect route for each message sent. Fig. 3.9 shows straight lines connecting the source and destination of each message. Note that these lines do not represent the actual paths followed by the messages and are used, instead, to illustrate the general message patterns. Each message was transmitted after a uniform random delay period.

We were interested in comparing our stochastic approach to geocasting with some existing alternatives. We specifically excluded all approaches which required that a node be designated as a router, because this approach is inconsistent with our own efforts at ad-hoc message delivery. Obviously, this would be an interesting comparison to perform at a later time. In each simulation, we configured the nodes to behave according to a distinct mechanism from among the six geocasting approaches of interest to us. These are listed below.

- Flood search
- Rectangle-based Location Based Multicasting (LBM)
- Distance-based LBM
- Voronoi region relaying
- Utility-based geocasting
- Utility-based geocasting with HELP flag.

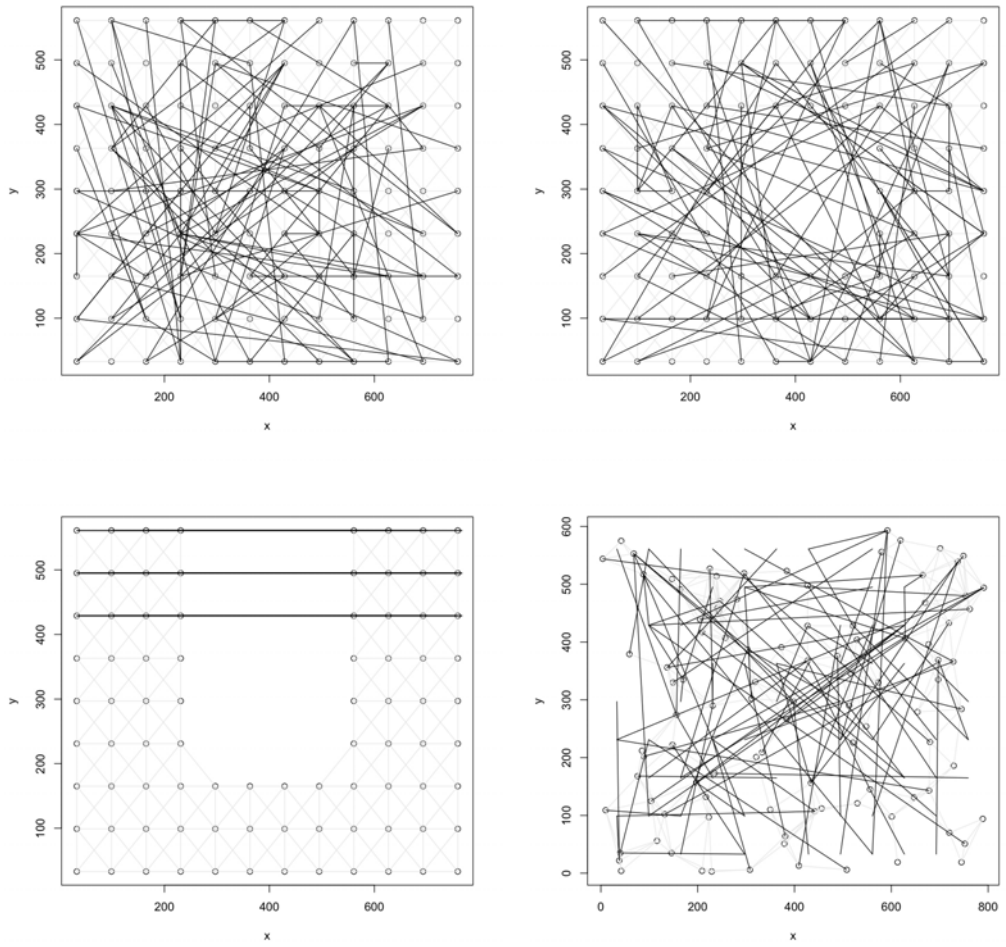


Figure 3.9. Lines connecting message sources and destination in each scenario.

Since the utility-based approach is stochastic, we ran 5 simulations for each of the two variants (with and without the HELP flag) and for each of the four topologies. We then used the average value of the parameters under consideration for the final results.

3.7.2 Experiment Results

After each scenario's simulation was completed, we collected a detailed account of each message transmitted. Specifically, we recorded the intended message recipients, which of those eventually received it, the total number of times the message was transmitted (i.e. its total cost), and the number of these transmissions that occurred after the message was

delivered to all nodes that were in the area of interest (i.e. wasted transmissions).

In the tables below, we list the performance of each of the six approaches to geocasting in our scenarios. We measure cost as the sum for all messages of the total number of times that a given message is transmitted. The minimum cost was determined by computing an optimum path from source to destination with full knowledge of the nodes' locations. The actual cost is that incurred by the protocol in use during the actual simulations. The waste is the number of times the messages were transmitted after they had already been delivered to all intended recipients. Lastly, the delivery rate is the number of intended message recipients who actually received the messages divided by the total number of intended message recipients.

The first scenario was run as a baseline to document the performance of all approaches under the most benign of conditions. The results are detailed in Table 3.1. As expected, all approaches were able to deliver all messages to all intended recipients. Had this not been the case, we would have had a strong indication that our implementation was incorrect.

Table 3.1. Scenario 1 (Regular) Results

Approach	Min. cost	Actual cost	Waste	Delivery rate
Flood	583	10800	4767	1.0000000
LBM Box	583	2808	377	1.0000000
LBM Distance	583	2924	324	1.0000000
Voronoi	583	4926	2521	1.0000000
Utility	583	3692	647	1.0000000
Utility w/HELP	583	3682	552	1.0000000

The results of the second scenario's simulation show that both versions of LBM did not fare as well as the other approaches under consideration. It is noteworthy that costs for all were expected to be lower because of the reduced number of nodes in the network. This reduction, as was previously shown in the top right of Fig. 3.8 is due to the presence of an exclusion zone in the center of the graph. The requirement for more circumvent routes that this obstacle imposes also drove up the minimum cost for message delivery. Table 3.2

presents the results of this scenario's simulation run.

Table 3.2. Scenario 2 (Concave) Results

Approach	Min. cost	Actual cost	Waste	Delivery rate
Flood	614	9600	4389	1.0000000
LBM Box	614	2470	330	0.9563492
LBM Distance	614	2387	271	0.9841270
Voronoi	614	4538	2341	1.0000000
Utility	614	3123	559	1.0000000
Utility w/HELP	614	3374	588	1.0000000

Scenario 3 was the pathological case intended to stress-test the various approaches under very adverse conditions. Unremarkably, all approaches experienced a substantial degradation in the delivery rates as is clearly seen in Table 3.3. The decreased values for cost and waste are predominantly due to the fact that so many delivery attempts failed, which prevented many nodes downstream from incurring the costs (and wastes) of relaying them. Note that the only mechanism that remains viable under these conditions is our utility-based approach with the use of HELP flags.

An interesting observation during the conduct of this experiment was that the nodes closest to the middle of the obstacle eventually depleted their batteries and ceased to participate in the message relay. This, in fact, expanded the obstacle's size. The next row of nodes seamlessly picked up the slack and started transmitting with significantly higher probabilities, which resulted in a consistent rate of successful message deliveries.

Table 3.3. Scenario 3 (Pathological) Results

Approach	Min. cost	Actual cost	Waste	Delivery rate
Flood	1359	8400	520	1.0000000
LBM Box	1359	431	0	0.0000000
LBM Distance	1359	1182	27	0.1220930
Voronoi	1359	873	0	0.0000000
Utility	1359	1255	44	0.1497093
Utility w/HELP	1359	3352	213	0.8154070

Finally, scenario 4 was intended to be more representative of real-world network topologies. As we discussed in section 3.7.1 above, nodes were randomly placed with the only restriction being that they must yield a connected graph. All geocasting approaches were able to deliver the majority of the messages to their intended recipients, albeit with varying degrees of success as is shown in Table 3.4.

Table 3.4. Scenario 4 (Uniform) Results

Approach	Min. cost	Actual cost	Waste	Delivery rate
Flood	528	9500	4538	1.0000000
LBM Box	528	1713	163	0.7289073
LBM Distance	528	2000	192	0.9170124
Voronoi	528	3210	1215	0.9820194
Utility	528	2431	334	0.9336100
Utility w/HELP	528	2714	429	0.9889350

3.7.3 Discussion of Results

Scenario 1, again, was a baseline test to see how the system performed under benign conditions. The one remarkable item was that the Voronoi approach required significantly more transmissions to deliver the same number of messages. In fact, this is a trend that is consistent in all scenarios for the Voronoi approach. Our analysis indicates that the reason for this is that the algorithm does not take into account whether or not the destination region was already reached by some other node's transmission. Instead, all nodes in the correct Voronoi region will attempt to relay the message, even when this has already been received by its intended recipients. This, in effect, can cause message retransmissions to circumvent the area of effect, oftentimes unnecessarily. Though a simple modification to the protocol could correct this, it would also hinder the mechanism under some circumstances. Accordingly, we decided to make a note of this and not weigh the cost of this approach too heavily. For now, we will be more interested in high delivery rates.

The second experiment, involving the concave topology of scenario 2, illustrates the

well-known shortcomings of the LBM approach in the presence of even moderate obstacles. This phenomenon was described earlier as empty forwarding zones. Still, the delivery rates remain fairly high, which means that the vast majority of messages did not experience this problem.

The pathological case represented by scenario 3 was of much interest to us, because it provides information about the behavior of all the approaches at what amounts to a boundary condition. Though we could have made the scenario even more adverse by simply increasing the height of the simulation area, we found that the current setup was sufficient to show the rate of decay of the geocasting approaches. The results from this simulation show that our approach, with the HELP flags discussed in 3.5, is the only approach that does not collapse in the worst case. This result confirms our assertion that our stochastic approach is viable and represents a substantial contribution to the body of knowledge of geocasting.

Lastly, the uniformly distributed topology of scenario 4 substantiates our assertion of our approach is at least as good as the Voronoi region one under normal conditions. In fact, the utility-based geocasting mechanism we propose shows better performance in all key parameters: total cost, waste, and delivery rates. Unsurprisingly, the LBM approach was the worst performer as this topology offers plenty of empty forwarding zones that effectively eliminate the chance of delivering a significant portion of the messages.

3.8 Conclusions

Our stochastic utility-based approach to geocasting is not only good, it is the only solution that works in the worst case scenario. This is an important consideration because we know that dynamic networks will change from fairly benign to very adverse and back again. If we want to ensure reliable message deliveries, the approach we use must perform well under the worst possible conditions. We have shown that Ancile does just that.

Furthermore, Ancile spreads transmission costs so as to maximize node lifespan. As

nodes in a direct communication path start running low on battery power, they start to relay with decreasing probabilities even as their neighbors with more battery power become relatively likelier to retransmit. When nodes disappear from the network altogether, whether due to depletion of their batteries or any other reason, neighboring nodes, again, pick up the slack. The result is a resilient framework that performs well in very challenging conditions.

CHAPTER 4

A Cooperative Spatial-Aware Shared Cache

4.1 Introduction

A shortcoming of our framework thus far, is that we still rely on connectivity to centralized data stores across an ad-hoc network that is frequently slow and unreliable. Nodes attempting to acquire situational awareness (SA) would likely experience delays and failed transmissions, particularly as they moved further away from their headquarters. These communications problems oftentimes translate into loss of life in military and emergency response scenarios. An attractive solution is to take advantage of neighboring nodes to satisfy some queries.

Our approach is that, though it would be very desirable to totally sever the links to a remote data server, this will prove to be impossible in many realistic scenarios. Instead, our efforts try to minimize the need to use those links and instead rely on the information known by neighboring nodes to satisfy some queries. The idea, which is sketched at a very high level in figure 4.1, is to have nodes take the following into account whenever they make a decision on whether or not to adopt a information item into their individual caches.

- The predicted length of the node's stay within the region

- The known contents of the neighboring nodes' caches
- The importance of the specific information item

Nodes are able to predict their stay within the region by monitoring their own location and their velocity vectors and comparing those with the region's boundaries. Depending on processing capabilities and available power, this calculation is made only upon entering the region or re-computed whenever a new information item arrives. Since we are primarily interested in dismounted personnel, the first option requires very little additional power considering the speed of an average person and the size of the regions.

The contents of a node's cache are known to other nodes, because they are advertised in the node's beacon signal. This beacon is transmitted periodically at low power to allow neighbors to be aware of each others' location, velocity, and cache contents. The beacon is described in more detail in section 4.2.2

When a node leaves the region, it jettisons its cached data thereby giving the remaining nodes a chance to adopt some or all of the items that would otherwise be lost. All neighboring nodes would receive the messages, and individually decide whether to adopt the data items into their own share of the collective spatial cache. The adopting nodes may have to make room in their own caches for the new items, which means that they have to periodically reevaluate the value of the items in their own caches in order to identify the least valuable ones for possible deletion. This process of jettisons and adoptions reduces the transmission and time costs of repairing the cache as nodes leave.

4.1.1 Motivational Example 1

A military patrol in a hostile zone suspects an improvised explosive device (IED) has been emplaced in their area. They report this threat up their chain of command so that the proper commander can dispatch an explosive ordnance disposal team to investigate. Though the central server is the data repository, the patrol members' mobile devices can act as caches for

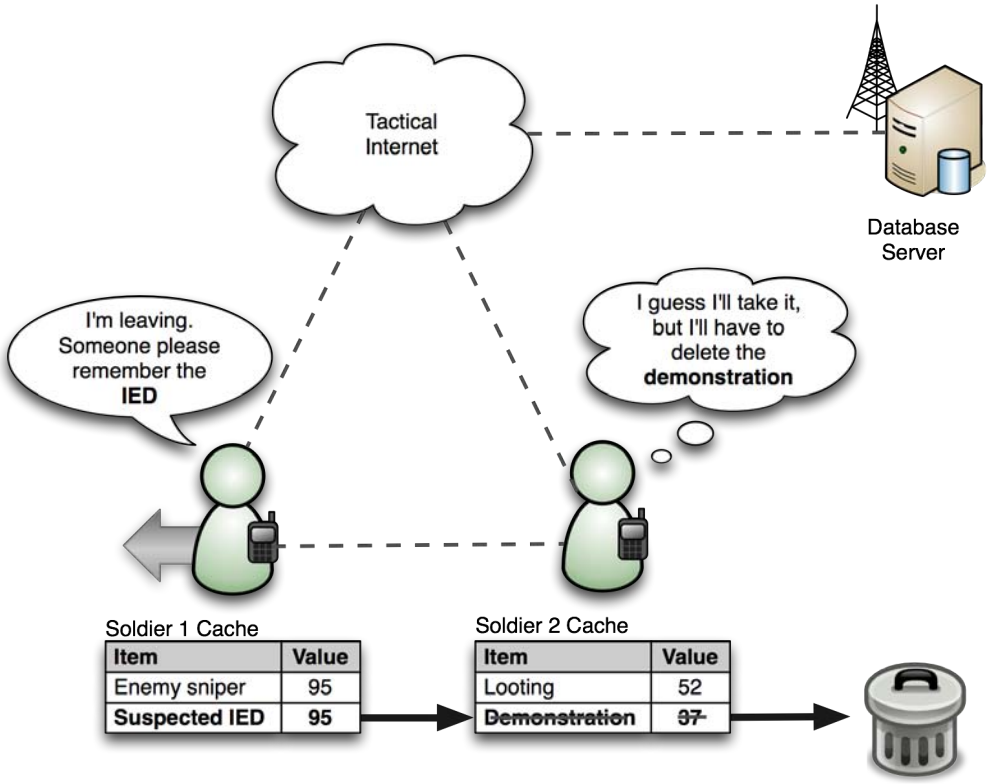


Figure 4.1. Cooperative Cache Scenario.

nodes in their vicinity. Furthermore, these caching nodes can detect the arrival of new nodes in their area and push the threat information to them. When the patrol returns to base, this cached data leaves with them, forcing friendly forces in the area to query the remote server explicitly until one of these nodes starts acting as a cache. If the patrol could handover these data to other nodes when they depart, the cost and speed of accessing them would remain low. This could literally mean the difference between life and death.

4.1.2 Motivational Example 2

A search team in disaster operation is combing a large area looking for survivors. The team finds a seriously injured survivor who needs immediate evacuation. The team leader reports their discovery to the rescue operations center, leaves a paramedic behind to stabilize the patient, and moves the rest of the team out in order to continue searching for others. Though the paramedic has a portable communications device, the report was filed by the team leader,

so only that device and the central server know of the casualty. If, as it leaves the area, the team leader's device could ask the paramedic's device to cache the report, then that data would remain in the appropriate spatial region. Later, as a truck drives by en route to a supply point, and it's on-board computer requests situational awareness on its location, it would learn from the paramedic's caching device of the location of the casualty. The reduced time required to get the patient to a medical facility could very well save a life.

The work we describe in this chapter could be developed into a significant enabler for sharing situational awareness. As such, it would allow personnel in both scenarios above to operate with greater effectiveness and efficiency, which could very well lead to saving lives in the situations of interest to us.

4.2 Architecture

Our shared spatial cache allows nodes to collectively cache all known data pertaining specific spatial regions. Under optimal conditions, this allows any query on that region to be answered using the shared cache. Clearly, this requires that there are enough nodes to cache all known data for that region. Our approach is spatial both in terms of the data it contains, as well as in terms of the location of the cache: a spatial region. As nodes enter that region, they become potential repositories for some of its data. When nodes leave the region, they hand whatever data they may be caching for that region to their neighbors; the neighbors, in turn, independently decide whether or not to adopt the data into their own portion of the shared cache.

4.2.1 Cache Regions

Most civilian geospatial applications use the Universal Transverse Mercator (UTM) to specify locations based on degrees of latitude and longitude. Though UTM is intuitive and allows

for arbitrary precision in specifying a point on the Earth, military land forces in the North Atlantic Treaty Organization (NATO) use the Military Grid Reference System (MGRS) instead. MGRS divides most of the planet into 100 km. by 100 km. grid squares, which are then subdivided into 10 km., 1 km., 100 meter, 10 meter and 1 meter squares. Unlike degrees of latitude and longitude, grid squares maintain consistent lengths, which makes them preferable for applications in which distances between regions are frequently determined.

The MGRS provides us with a convenient way to define the spatial regions that define our caches. Their terse naming convention has the added benefit of reducing the space required to identify regions and data within them. Furthermore, the fact that MGRS is widely used in virtually all military as well as many emergency operations makes its use justifiable from the users' perspective.

We divide space into regions using MGRS. Note that any similar regular decomposition of space (e.g., degrees of latitude and longitude) will work equally well, provided that the total area of a region is not excessive with regard to the amount of memory in the mobile devices. For ease of tractability, we will only simulate a single cache region in section 4.4. However, it makes sense to allow nodes to participate in the shared caches of multiple regions since the nodes are expected to move fairly frequently. The number of regions in whose caches a node may participate is bounded by the information density (i.e., number of data items) of those regions and by the amount of available cache memory in the nodes themselves.

If a node is configured to participate in multiple shared cache regions, it will choose the regions that are closest to it. A node always participates in the cache for the region in which it is located, which we call its home region. Every region that touches the home one is then examined in turn to see the minimum distance between the node and the closest point in that region. In the case of square regions such as those described by the MGRS, eight regions border the home region, so eight minimal vectors are calculated that touch each of those neighboring regions. The shortest $r-1$ vectors are selected for a node that participates

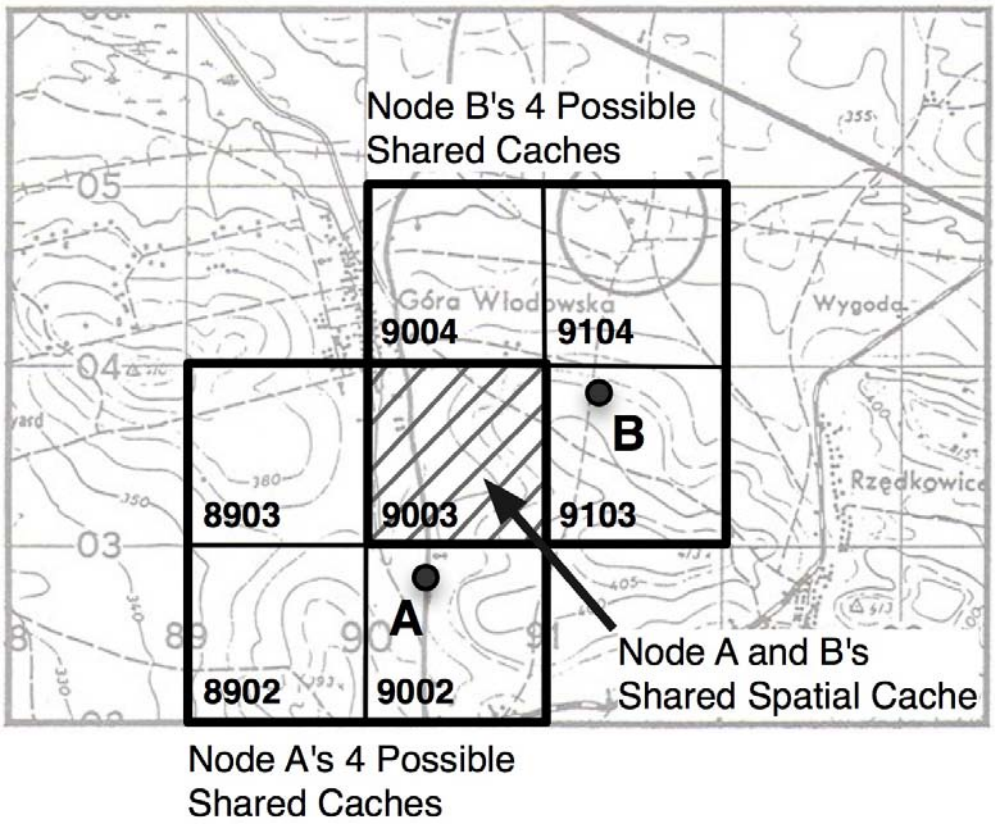


Figure 4.2. Shared Spatial Cache Regions

in r shared caches. The regions to which these vectors point are then added to the node's collection of caches.

In figure 4.2, for instance, we depict four shared caches ($r = 4$), each based on 1,000 x 1,000 meter grid squares. In this configuration, nodes participate in the four regions whose closest points to the node are minimal. This is to say that a node would participate in the shared cache for the region in which it is currently located, and also in the shared caches for the three neighboring regions whose closest point is closest to the node. Node A, in figure 4.2, participates in shared caches for grid squares 8902, 9002, 8903, and 9003. Similarly, node B only participates in shared caches for grid squares 9003, 9103, 9004, and 9104. Since the only common shared cache between nodes A and B is the one for square 9003, this cache will be the only one in which the two nodes will collaborate with each other.

As nodes move in space, the shared caches in which they participate change too.

Suppose that, in the example depicted in figure 4.2, node A moved one kilometer due north along the road on which it is depicted. Then it could no longer participate in shared caches for squares 8902 and 9002 and it would, instead, be able to do so for squares 8903, 9003, 8904, and 9004. If that were the case, then it would then collaborate with node B on the shared caches 9003 and 9004. If node A continued moving another kilometer up the road, it would no longer participate in the shared cache 9003, and would jettison any items from that cache so that other nodes such as node B could cache the data instead.

4.2.2 Beacons

Our framework depends on nodes being aware of each others' effects on the shared cache. Specifically, a node is interested in knowing who its neighbors are, where they are, where (and how fast) are they moving, and what entries are contained in their caches. This information is exchanged through the use of periodic beacon packets that are broadcast at low power by each node. The frequency of the beacon transmission is inversely proportional to both the node density in the region (to reduce congestion) and the available power in the node's power source (to enhance its longevity). Figure 4.3 describes the structure of a beacon packet.

4.2.3 Cache Structure

Assuming that a node is interested in participating in the shared caches for r regions, then its own cache consists of $r + 1$ lists of pages: one for each of the r spatial caches in which the node may collaborate, and one for all free pages that are available for use. Initially, the cache lists are empty and the free list contains all the empty cache pages for the node. As items are adopted for a spatial cache, pages are removed from the free list and placed in the corresponding cache list. When a node moves away from a spatial cache region, all pages in that cache's list are moved to the free list. Note that the contents of the pages are not immediately deleted, which means that if the node subsequently moves back into that region

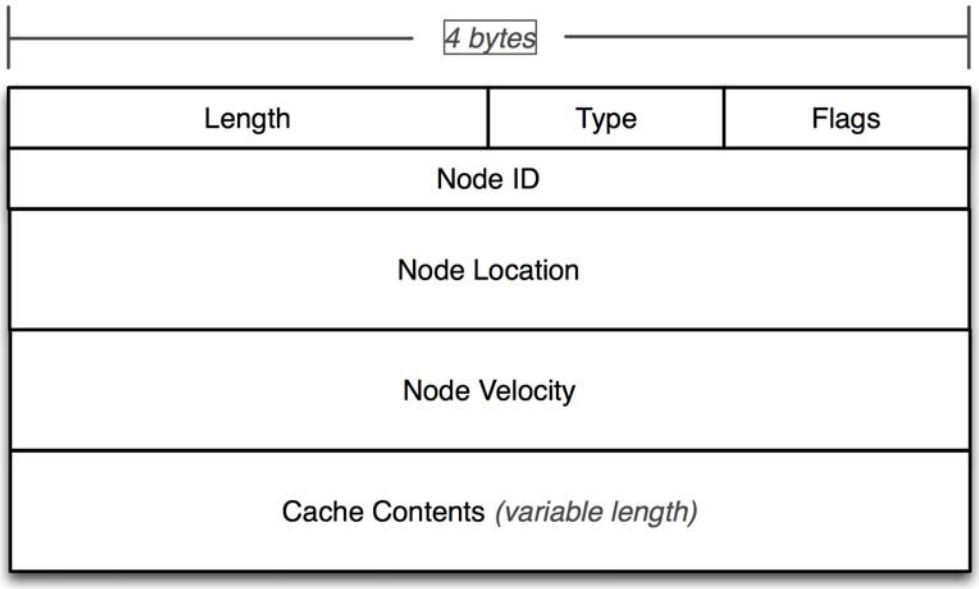


Figure 4.3. Beacon packet

it may be able to restore some or all data previously cached, as long as the pages were not already used for and overwritten in another cache.

Each page has pointers to the next and previous pages on its list, and is divided into one or more slots. Each slot in turn contains one data item from the shared spatial cache. The slot also contains metadata used for cache management. Specifically, each slot has fields for the following metadata.

1. Authoritative server ID: this is the unique identifier of the server from which this data item originates.
2. Number of cache hits: the number of times that this item has been used to respond to a query (either from a neighbor or a local application).
3. Number of cache copies: the number of nodes that are participating in this spatial cache and that are known to have a copy of this item. This value includes the local node, so it is always greater than zero.
4. MBB: the minimum bounding box for the data item, obtained from the item's R-tree index entry.

5. Last update: the time at which this the item was last updated (or created).

6. Expiration: the time after which the item is assumed to be stale and, thus, unreliable.

When nodes exchange information items, they transmit the entire contents of the cache page slot: data and meta-data. This ensures that adopting nodes will have access to the item's metadata for inclusion into their own cache pages. Adopting nodes increment the number cache copies, but retain the values in the other fields.

4.2.4 Building a Shared Cache

Suppose a node arrives at a region and detects no neighbors. Since nodes are assumed to automatically acquire relevant (to the user's role) information about any regions through which they traverse, then this node will query an authoritative server for whatever data it needs for this region. The mechanism by which this message is routed is the subject of Chapter 3 of this thesis. With that query, the node sends a parameter specifying the number of regional items it is willing to cache. The server eventually responds to the query and may provide, in addition to the requested items, extra ones up to the value specified by the requester in its cache size parameter. In addition, the server will send a count of the total amount of data items that it has for the region. Note that, since the query result may be a subset of the total data for the region, the total data item count may be different from the number of items returned. Let's say that the server responds with more items than the node can store in its cache. The node will pass all the data up to the application layer for processing and keep some of it (as much as it can) in its cache so that it may be used to respond to later queries from either the local node or others nearby. The node will also make a note as to how many total items reside in the server for the region in question.

Suppose another node subsequently arrives. Again, it will attempt to gather all relevant information by querying neighbors or servers. Its only neighbor (the one in the preceding paragraph) may respond with the relevant data items from its cache if it has any,

as well as the number of total items that exist in the authoritative server. Now the second (querying) node may be able to immediately provide *some* relevant data to its applications even as it waits for the remaining items from the server. When the rest of the items arrive from the server, the querying node will pass them up to its applications, and then add to its cache as many items as it can from among the set that was not cached by its neighbor. Since items cached by its neighbors are only one hop away, the node will not also cache them. This means that in the early stages of a shared spatial cache, data items will tend to exist in only one node's cache. Though we need the replication of data items among multiple nodes in order for the shared cache to survive the loss of a portion of its constituent nodes, this is a feature that emerges later in a cache's lifespan.

Cached items are tracked approximately by the server. In order to minimize communications costs, the server does not receive feedback from the nodes as to which items are collectively cached in the region. Instead, the server marks each item with a timestamp indicating the last time it was submitted to a shared regional cache. In this manner, the server is able to cycle through its items as it chooses which ones to send to the region for caching, ensuring that the oldest ones are refreshed first as new nodes request non-cached data and indicate a willingness to host additional data in the cache.

As additional nodes arrive in the region, each node will come to know what data is available in its neighbors' shared caches as well as how much total data exists for the region. Since messages are broadcasted, other (existing) nodes in the region will also become aware of the shared cache contents of each other, which helps them determine which items are safer to delete when they start running out of space in their shared caches. As the node density in the region increases, the server eventually cycles through the items pertaining to the region and additional copies of previously-cached items emerge within the cache. Furthermore, as nodes depart the region, they jettison their cached items and this, as we discuss in detail in section 4.2.5, commonly results in multiple copies of cached items. Note that a node will not cache multiple copies of the same item, but different nodes in the region may, over time,

hold copies of the same item.

The cache admission process is described in detail in algorithm 4.1. We require that the length of a node's cache at any point in time is no more than the stated maximum length for that cache. We use this precondition in lines 3 and 12 to test for a full cache. Lines 1 through 6 drive nodes to always cache data arriving from a server in response to a query by that node. The rest of the algorithm describes the random process by which nodes decide whether or not to cache items from either a server or a jettisoning neighbor that are not specifically sent to those nodes. Note that the formula for p in line 9 is derrived explicitly in section 4.3.2 later in this chapter.

```
CACHE-ADMISSION(msg)
 1  if (msg.src = SRVR and msg.dst = THIS)
 2     then
 3          if (cache.length = cache.maxlength)
 4             then
 5                  JETTISON(msg.payload.size)
 6          cache ← cache + msg.payload
 7  elseif (msg.src = SRVR or msg.type = JETTISON)
 8     then
 9          p ← (7.41n + √(50.9n^2 − 21.64n)) / (2n^2 + 10.82n)
10          if (RANDOM() ≤ p)
11             then
12                  if (cache.length = cache.maxlength)
13                     then
14                          JETTISON(msg.payload.size)
15                  cache ← cache + msg.payload
```

Figure 4.4. Cache Admission Algorithm

4.2.5 Data Jettison

Our framework a jettison mechanism by which nodes, as they depart the shared cache region, eject the contents of their caches. The purpose of this process is twofold: firstly, it serves to keep high-value items within the cache even as the nodes that hold them leave, and secondly,

it serves to create multiple copies of popular items within the cache, ensuring that the loss of any one node does not adversely affect the cache performance.

The jettison process is illustrated in figure 4.5. It depicts a spatial region with 26 data items $\{a, .., z\}$ and four nodes A, B, C, and D. Node A is crossing the region boundary and jettisons its two cached items, b and c. The messages containing the items also let recipients know the number of neighbors that node A had within the region just prior to leaving it. This information lets other nodes determine the expected number of nodes that will consider adopting the item. Intuitively, the more neighbors a departing node has, the lesser the chance that any one of them will adopt the jettisoned item. If we chose this probability carefully, we can ensure that only a relatively small number of nodes adopt the jettisoned item, which helps distribute data throughout the cache so that the loss of no single node results in drastic loss of information.

Node C has empty slots in its cache for this region and independently determines whether or not to adopt the jettisoned items based on the probabilistic model mentioned in the preceding paragraph. In the figure, we show node C adopting both items b and c. Node B, on the other hand, already has a full cache for this region and, should it choose to adopt one or both jettisoned items, must make room for them by discarding one or two of its previously cached items. In the illustration, node B chooses to adopt item b only. In order to make room in its already full cache, node B decides to discard item a, in part because it knows that node C is already caching it. Recall from our earlier discussion that nodes monitor the broadcast communications medium to discover, among other information, the contents of their neighbors' caches.

Note that the number of replicas of item b increased by one as a result of node A departing the region while, simultaneously, the number of replicas of item a decreased by one. This makes sense because our jettison process deletes the least valuable items in adopting nodes in order to make room for the items jettisoned by departing nodes.

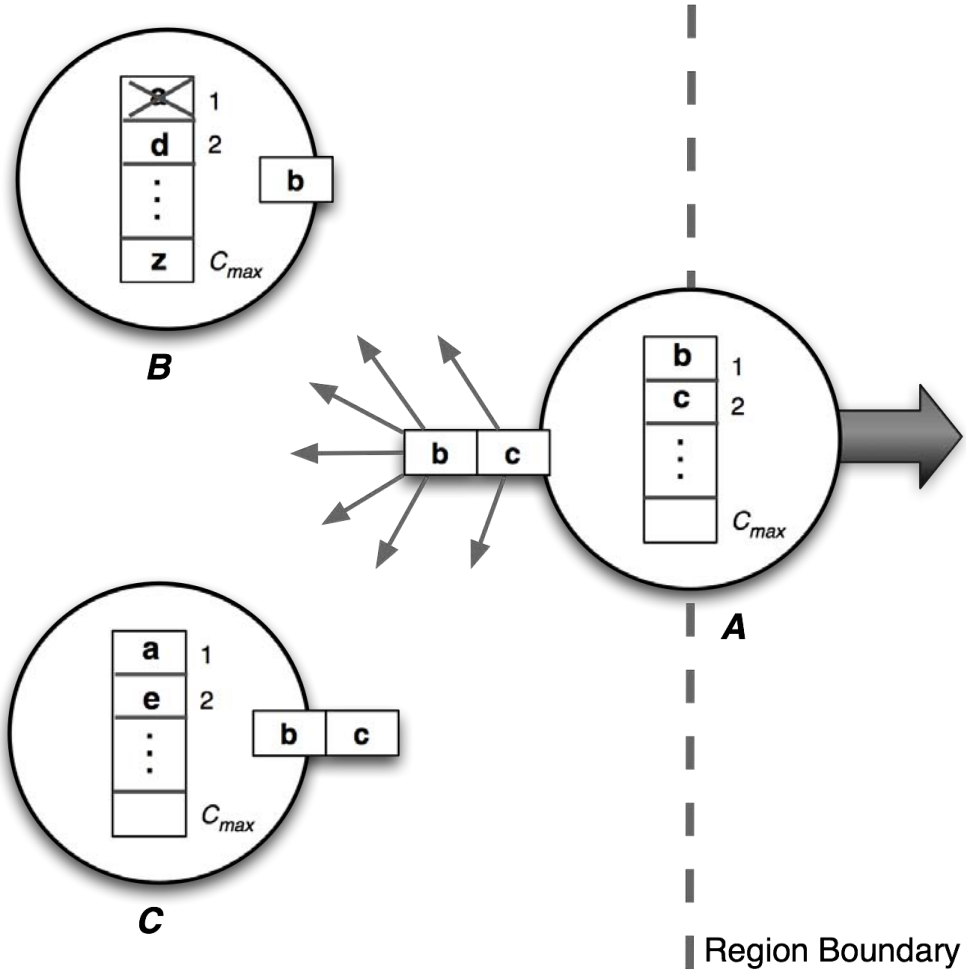


Figure 4.5. Data Jettison

4.2.6 Cache Replacement

When choosing items to discard in order to make room for new item adoptions, nodes calculate the relative value of the items they already cache. This value is a proportional to the number cache hits (h) for the item which is a measure of popularity, and to the inverse of the number of known replicas ($\frac{1}{r}$). Note that r is guaranteed to be at least one, since the local node has a copy of the item. The term α allows us to control the relative importance of these two factors in deciding the value of a given item. Therefore, we chose to assign this value using the formula in equation 4.1.

$$v(i) = \alpha \frac{h}{r} \tag{4.1}$$

Though we choose to use only the number of cache hits and replicas in our value function, many other possibilities exist which could be explored at a later time. Among these, we have identified the following.

- Distance to the server which implies a higher transmission cost
- Size of the item which also requires a higher transmission cost
- Priority
- Time-To-Live, since an item could be considered less valuable if it is about to expire

A value is computed in accordance with equation 4.1 for each item in the cache. The item with the lowest value is replaced with the newly adopted item. In this case, the data from the previously cached item is lost.

The cache replacement, or deletion, process is described in algorithm 4.6. Note that we examine each item in the cache once to determine its value. As we do this, we keep track of which item has the lowest value so that, once the loop terminates, we know which item to

```
REPLACE()
 1  choice ← 0
 2  minValue ← ∞
 3  for (i = 1 to cache.length)
 4        do
 5           d ← distanceTo(cache[i].server)
 6           h ← cache[i].hits
 7           r ← cache[i].replicas
 8           v ← h/r
 9           if (v ≤ minValue)
10              then
11                   minValue ← v
12                   choice ← i
13  if (choice > 0)
14     then
15           delete(cache[choice])
```

Figure 4.6. Replacement Algorithm

delete. Note that the test in line 13 is necessary for the case in which the size of the cache is zero.

In general, nodes do not necessarily delete the data from a shared cache when they leave its region. If there is no new data of higher value, nodes could conceivably carry around the old data indefinitely, or until the items have to be replaced with newer ones. Upon re-entering the region at a later time, they could conceivably re-activate old items that have not expired or been deleted.

4.2.7 Cache Consistency

Under normal conditions, new or updated data items are automatically relayed to the affected region by the network. This process is described in in Chapter 3. This means that the data in the spatial region is updated as needed without any need for action from the nodes within it. This mechanism can break down whenever the network becomes disconnected from the servers, from the data sources, or from both. Under those adverse conditions, we need a mechanism for ensuring that stale data is identified as such and, if appropriate, removed

from the shared cache.

The absence of updates on a known data item may mean that there is no fresher information on it, but it can also mean that connectivity to the server has been broken. The nodes, however, do not know which is the case and must consider the item to be stale. When a node receives a query which may be answered with a stale item it has, it will first issue a query to the server that owns the data, and then respond to the neighbor node with the stale data. The neighbor will realize that the data is stale, but may use it while fresher data is acquired. Once the caching node receives a response to its query to the server for the fresh data, it broadcasts it within the region so that other caching nodes can update their own copies.

Nodes do not proactively refresh data from the server prior to it becoming stale. Recall that, under normal conditions, this data will be automatically delivered to the region by the server or an updating node. Under this assumption, it would be wasteful to spend precious CPU cycles searching through the cache for items that are about to become stale. Furthermore, as we mentioned in section 4.2.4 above, the server may refresh items as it responds to new queries using cache item timestamps.

Our approach requires the data creators and updaters to provide an estimated expiration time for their data. In some cases, such as permanent structures (e.g., buildings), there is no expiration time. In other cases, such as a moving person, the expiration time is simply an estimate of how long the data creator or updater thinks the person will remain near their current location. Obviously, this later case has a very large margin of error. Individual nodes can calculate how dynamic a given spatial data item is by simply comparing its last update time with its expiration time. The shorter this interval is, the more dynamic the object.

4.3 Analytical Models

It was important for us to understand the conditions under which nodes would score misses against the spatial cache. Clearly, misses will occur whenever the data on a spatial region exceeds the aggregate cache capacity of nodes in that region. What was not all that clear was how the hit ratio behaved as a function of node density and mobility.

4.3.1 Queue Model

To help us understand the behavior of our scheme, we modeled a region as a $M/M/\infty$ queue, as shown in figure 4.7. We assumed that nodes arrived at the region at a rate which followed a Poisson distribution with rate λ. The nodes would loiter or remain in the region for an amount of time following an exponential distribution with mean $1/\mu$. This means that the expected number of nodes in the region is $\bar{n} = \lambda/\mu$ and the rate at which nodes leave the region is $n\mu$ [46].

We assume that nodes are connected to all other nodes in that same region using a shared broadcast medium with usable bandwidth B. If all data items have identical size S, then cache entry transfers are bounded from above by B/S. This means that, even under ideal conditions, we cannot transfer more than B/S cache entries per unit time.

So the question is, how many entries can each node have as part of the shared spatial cache? Using the derivations above, it follows that the maximum number of cache entries c_{max} must satisfy inequality 4.2.

$$c_{max} < \frac{B}{n\mu S} \tag{4.2}$$

Now that we know the maximum contribution of any one node to the shared cache, we want to understand the probability that every data item concerning the region is cached

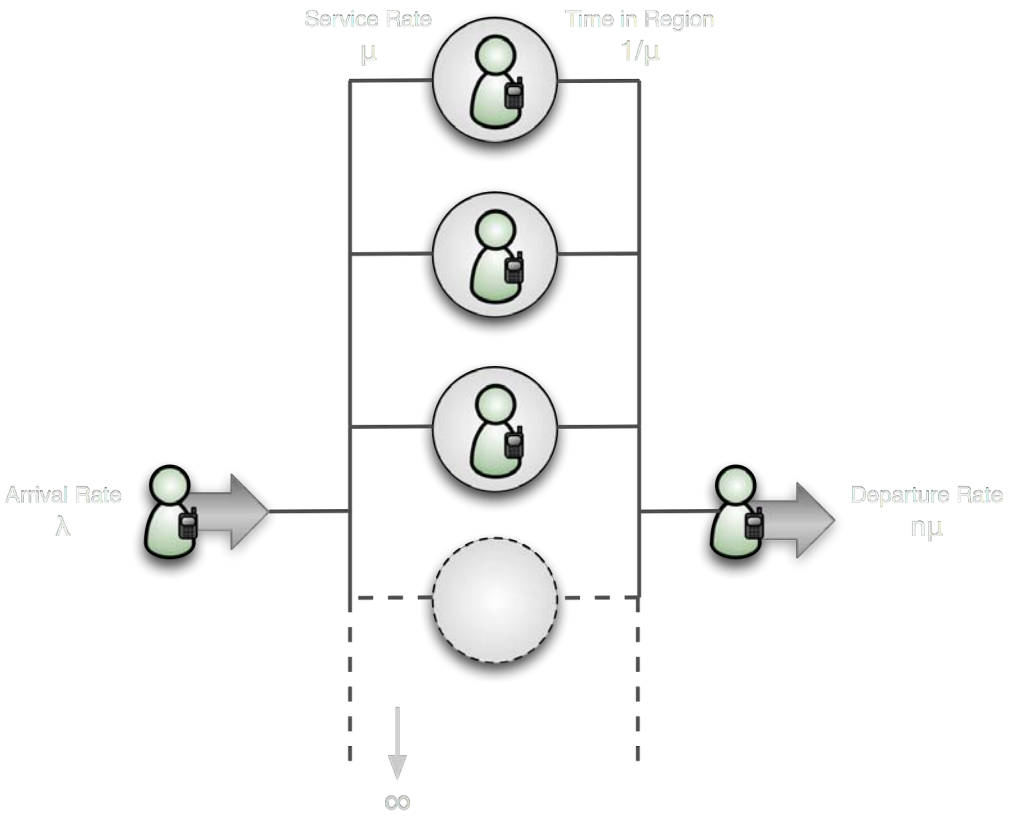


Figure 4.7. Spatial Regions as $M/M/\infty$ Queues

by at least one node in that region. To do this, we first have to define the probabilities for the data cached by a departing node to be successfully transfered to another node that is still in the region.

4.3.2 Individual Cache Model Analysis

When a node is about to leave the region, it will transmit (or jettison) it's shared spatial cache entries so that they may remain within the region. Since the items are sent out without waiting for an acknowledgement of receipt, we want to ensure that there is a reasonable chance of some other node picking them up. The departing node helps the remaining nodes in this effort by letting them know, in that final message, how many neighbors it has.

When a node receives a jettisoned cache item, it will determine the probability of storing it as a function of how many neighbors the departing node had. If the number is large, then each receiving node should be less likely to store the item, since some other node may very well do it. This dynamic probability calculation makes it more likely that at least one node (but not too many) stores the jettisoned item. We can then model this situation using a binomial probability distribution where the number of experiments is the number of neighboring nodes n, and the probability p that each node stores the item is unknown. The question is what value of p will give us a 99% probability that at least one success will occur?

To answer this question, we first note that the mean number of successes is np and the standard deviation is $\sqrt{np(1-p)}$. We need a p such that, with probability 0.99, at least one node will cache the jettisoned data, or $P(X \geq 1) = 0.99$ (where X is the number of nodes caching the data). This is the same as saying $0.01 = 1 - P(X < 1)$. We can then use a normal approximation $X \sim N(np, \sqrt{np(1-p)})$, and derive equation 4.3.

$$P\left(X < \frac{1-pn}{\sqrt{np(1-p)}}\right) = 0.01 \tag{4.3}$$

We can use basic probability tables to assert that the 0.01 quantile of a normal distribution is -2.326, which means that, in order to compute p, all we need to do is solve equation 4.4 for p. After a bit of algebra, we are left with a formula for calculating the value of p as a function of n, which is described by equation 4.5.

$$\frac{1-pn}{\sqrt{np(1-p)}} = -2.326 \tag{4.4}$$

$$p = \frac{7.41n + \sqrt{50.9n^2 - 21.64n}}{2n^2 + 10.82n} \tag{4.5}$$

As we show in figure 4.8, we are able to achieve a very high probability that at least one node will adopt the cache item jettisoned by the departing node. At the same time, we see that the probability of any one node adopting the item drops sharply as the number of neighboring nodes increases. Despite this, as we show in figure 4.9, the expected number of nodes adopting the item is usually more than one.

Now that we know the upper bound on the number of data items in a node's shared spatial cache, C_{max}, and we know the probability that a node will adopt a new, offered data item, we are ready to calculate the expected number of items in a node's portion of the shared spatial cache. Recall that we need that number to be bounded from above by C_{max}.

From our earlier queue analysis, we know that the duration of a node's presence in a given spatial region is a random variable $Y \sim Exp(1/\mu)$, we know that the expected duration of a node's stay in that region is $1/\mu$. We also know that the expected number of nodes in

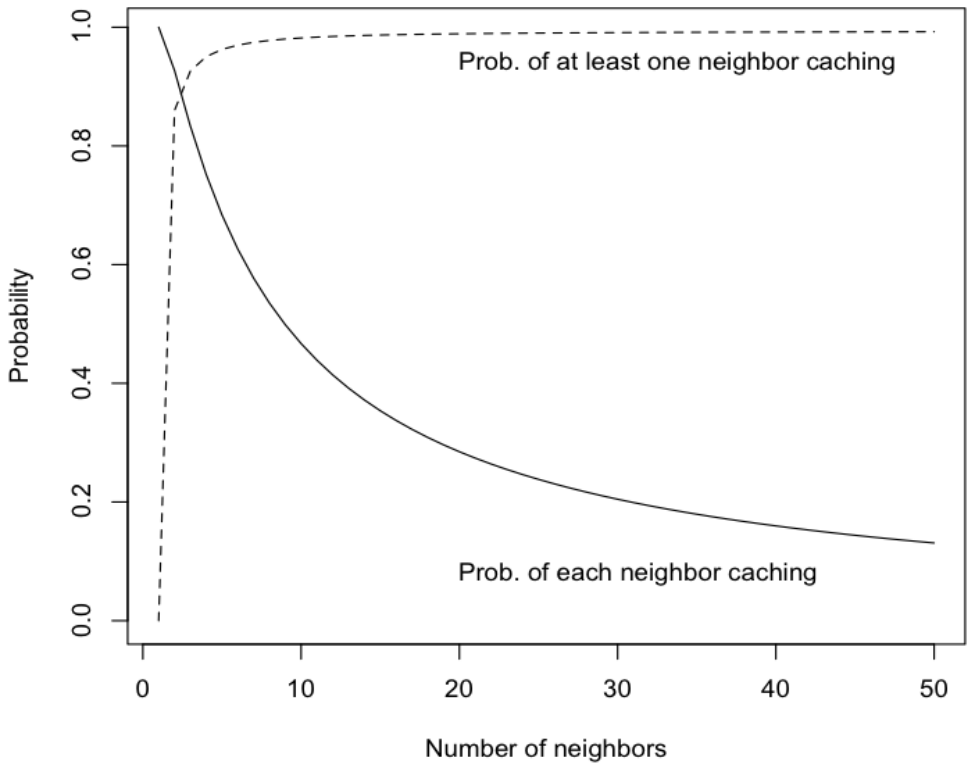


Figure 4.8. Probability of item adoption

that same region is λ/μ, and that the rate at which nodes will leave the region is $n\mu$. This means that the expected number of departures, and hence whole-cache jettisons, that a node will experience is μ^2/n.

By substituting the expected number of nodes within the region (λ/μ) in equation 4.5 above, we then know that the expected probability for adopting a jettisoned cache item is given by equation 4.6.

$$Ep = \frac{7.41\frac{\lambda}{\mu} + \sqrt{50.9\left(\frac{\lambda}{\mu}\right)^2 - 21.64\frac{\lambda}{\mu}}}{2\left(\frac{\lambda}{\mu}\right)^2 + 10.82\frac{\lambda}{\mu}} \tag{4.6}$$

If we consider the adoption of a jettisoned data item by each of $n = \lambda/\mu$ nodes as a series of Bernoulli experiments, then the number of nodes that adopt the item can be mod-

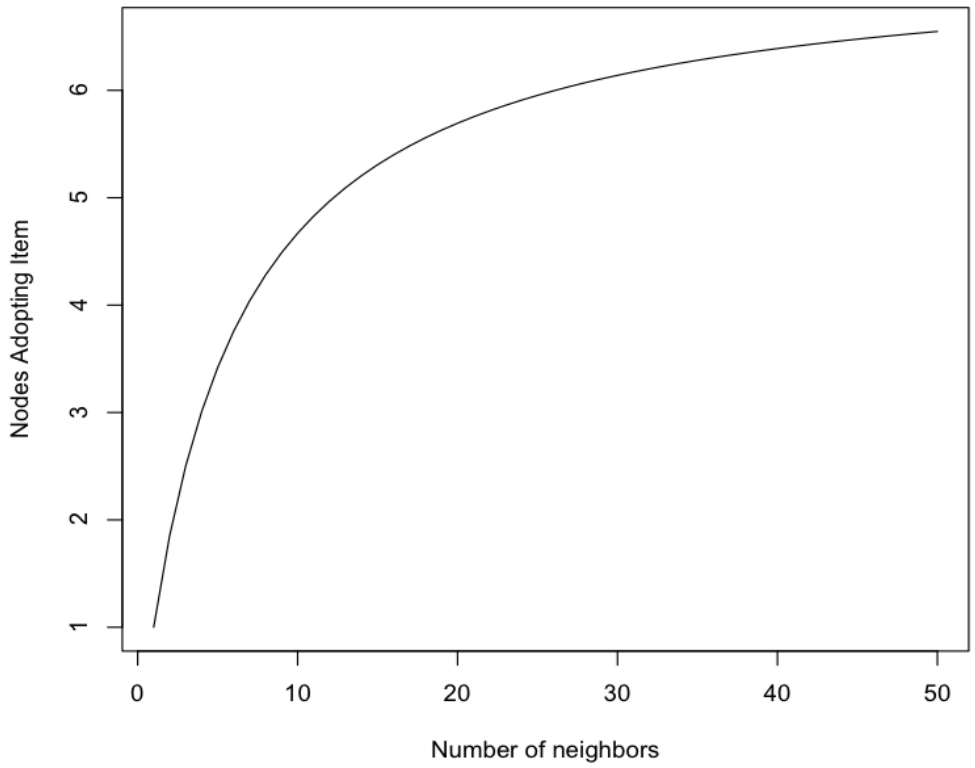


Figure 4.9. Number of nodes adopting an item

eled as a random variable A with a Binomial distribution $A \sim Bin(\lambda/\mu, Ep)$. The results for various values of n are depicted in figure 4.9.

4.4 System Simulations

4.4.1 Measures of Performance

The quintessential measure of performance in a study of caching performance is how often node queries can be addressed by the cache. This is commonly referred to as the *cache hit ratio* in database parlance. Our model predicts that the popularity of an item will lead to more nodes adopting it, so we are also interested in measuring the popularity of facts in node queries and the number of cached copies of each of those facts. Finally, we want to see how effectively the node caches are being used, so we need to measure the number of cached

items whenever a node leaves the region. These measures of performance are detailed below.

- Cache hit-ratio
- Number of copies of each item in the shared cache
  - Mean value
  - Relative to item popularity in query results
- Utilization of a node's cache upon exiting the region

4.4.2 Estimating Queue Model Parameters

Our entire system model hinges on the parameters of its underlying queue model. The arrival (λ) and service (μ) rates determine the number of nodes in the region (n_t) as well as the number of nodes departing the region (d_t) at any given point in time. Recall that the service rate μ is the inverse of the expected time a given node will remain within the region. The number of departures drive the number of cache data jettisons (j_t). These events, together with the number of remaining nodes, in turn tell us how many shared cache items reside at each node at a point in time ($N_{i,t}$). What we now need to know is what fraction of the total D data items for the region are actually in the shared spatial cache over time. The higher this value, the better our system's performance can be expected to be.

Our models quickly become analytically intractable as the number of interacting nodes increases. We therefore turned to simulations to get the answers we needed. Our first order of business was to determine reasonable representative values for our node arrival and service rates λ and μ. For this, we made use of BonnMotion, a mobility scenario generation and analysis tool developed at the University of Bonn [13]. With it, we created a total of 120 different mobility scenarios using two models and two population sizes, all operating in a square area measuring three kilometers on each side. The central, one square kilometer, region of this area is the one of interest to us. The first model used was the random waypoint mobility model [6], which is commonly used for modeling unconstrained pedestrian traffic.

Table 4.1. Queue Model Parameters

	λ		$1/\mu$		n	
Model	**mean**	**stdev**	**mean**	**stdev**	**mean**	**stdev**
Random Waypoint (30 nodes)	0.007149306	0.0003064346	0.001037162	4.965799e-05	6.720866	0.3400475
RPGM (30 nodes)	0.006862805	0.0007652106	0.001212537	9.288928e-05	6.508561	0.6059559
Random Waypoint (100 nodes)	0.02302894	0.003063120	0.001051836	3.529071e-05	21.46979	2.877443
RPGM (100 nodes)	0.0235813	0.001090551	0.001117267	5.86594e-05	22.66135	1.197213

We also used the reference point group mobility model (RPGM) [18], which allowed us to explore the effects of nodes traveling more or less in groups. For each of these, we chose 30 and 100 nodes in order to approximate the sizes of an infantry platoon and company (respectively), which are military units that could reasonably be expected to be operating in an area of nine square kilometers. All nodes moved at randomly chosen speeds between zero and five kilometers per hour to simulate people walking on foot.

We generated 30 distinct scenarios for each of the four cases under study and ran the simulations for eight hours each. The computations were done in R, and the the script we used is available in Appendix A.1The results, which are summarized in table 4.1 show that the arrival and service rates for the random waypoint and RPGM models were fairly similar for a given population. This is convenient, because it allows us to assume that the parameter values are independent of the mobility model, and thus allows us deal with fewer test cases.

Our simulations allowed us to revisit equation 4.2 and determine a reasonable approximation of it under the conditions in which we are interested. We wanted to choose values for the various parameters that were realistic, and yet accounted for conditions that were not benign. Accordingly, we chose the Random Waypoint and RPGM models with 100 nodes.

Table 4.2. Estimated Values of C_{max}

	Page size (in bytes)			
Bandwidth	**1,024**	**2,048**	**4,096**	**8,192**
384 kbps	1,562	781	390	195
1 Mbps	4,069	2,034	1,017	508
11 Mbps	44,759	22,379	11,189	5,594
53 Mbps	215,657	107,828	53,914	26,957

This allowed us to approximate $\lambda = 0.023$ and $\mu = 0.001$. For the value of n, we chose the upper range of the 99% confidence interval for that parameter, which by the central limit theorem turns out to be $n = 30$ in our experiments. Based on these approximations, table 4.2 gives us the maximum size of any node's portion of a shared spatial cache under common bandwidths and page sizes.

4.4.3 Shared Cache Model Simulations

So far, our analysis has only looked at the behavior of individual nodes. To understand the aggregate node behavior, however, we need to expand our probabilistic analysis so that we may look at the behavior of a region, and not just of the nodes within it. Specifically, we are interested in determining the hit ratios on the shared cache as a function of the size of a single node's contribution to that shared cache. In other words, given that each node contributed a specific number of items to the cache, and also given a fixed total number of items in the database server for that region, how often are node queries satisfied by the shared cache, as opposed to by the server.

In our simulations, we consider the same one-square-kilometer region that we studied

in the previous section. Again, we use the same mobility scripts based on the random waypoint model with a population of 100. Recall that the total area being simulated measures nine square kilometers, but we only track the nodes that enter the central square kilometer region.

We make a simplifying assumption that nodes are completely connected to each other within the region, so that any node is a single hop away from any other node within the same region. This allows us to focus on cache dynamics and not network transmission issues. We also assume that nodes will generate only one query per visit to the region, which is not all that unusual in the scenarios of interest to us.

4.4.4 Experiment Design

Our simulation environment is R, which is widely used in many sciences as well as in industry for simulating stochastic processes and for statistical analyses. We wrote a customized script that reads the mobility files from our earlier studies and translates them into a list of nodes that are present within the region of interest at each of the 28,800 seconds of our simulation. We developed a second script which takes as input these lists and simulates behavior of each node as it enters the region, becomes part of the shared cache, and eventually leaves the region, jettisoning any cached items.

We are interested in random vectors comprised of the items being requested by nodes as they enter the region of the shared cache. We approximate the values of these vectors using a Beta distribution with parameters $\alpha = 2$ and $\beta = 8$. The random value generated by our distribution is then multiplied by the number of items in the database server to yield the number of items in a query. For each element of the query response, we again use the same distribution to determine the specific data item identity. The Beta distribution allows us to model a situation in which we estimate that most queries will involve roughly 20% of the available data. This figure has been used before to simulate database queries. The Beta

distribution also accounts for rare, but important, queries which involve almost all data for the region.

Another random variable of interest to us is the vector of items being jettisoned by nodes as they leave the region and which are adopted by neighboring nodes. As we've already shown, this variable follows a binomial distribution. The probability associated with this distribution is discussed in detail in section 4.3.2 of this paper.

The main goal of our simulations was to determine the effectiveness of our proposed shared spatial cache. Specifically, we wanted to know the effects on cache hit ratios of varying local node cache sizes. Clearly, there are infinitely many ways of comparing local cache size with total number of data items for a given region, so we arbitrarily chose a 300 megabyte (MB) dataset for the region. We then ran the simulation repeatedly for node cache sized of 15 (5% of dataset) to 150 MB (50% of dataset) in 15 MB increments. Recall that we are using a total population of 100 randomly moving nodes of which no more than 30 are in the region at any one time.

For each of the chosen cache sizes, we ran the simulation using a local cache in which nodes do not collaborate with each other, as well as full regional node collaboration with a replacement policy based on our value function described in section 4.2.6. Additionally, for the shared spatial cache, we ran the simulation with and without the use of the jettison mechanism in order to compare its performance with and without this implemented. In this set of experiments, our principal measure of performance was the cache hit ratio. Obviously, a higher value of this metric is preferable, particularly if it does not require more memory than the alternative approaches.

Finally, we gathered statistics to show us the resiliency of the cache. Our measure of performance was the number of copies of a data item within the cache. As the number of copies in the cache increases, we can afford to loose a larger number of nodes without suffering cache degradation in terms of hit ratios. This is important when a catastrophic event such as an enemy attack destroys a significant fraction of the nodes in a region.

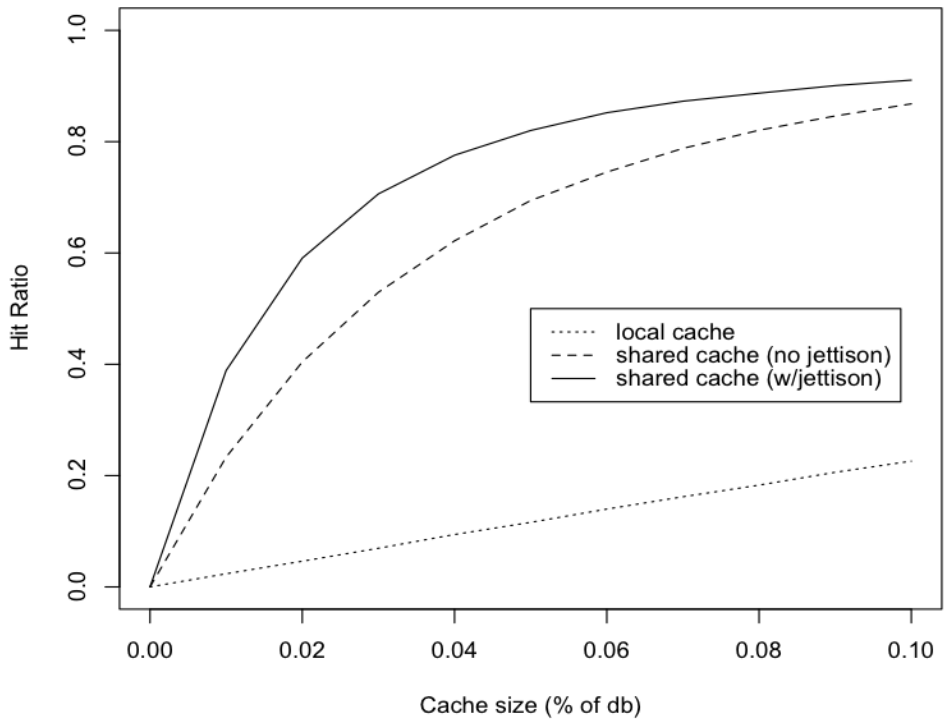


Figure 4.10. Cache Hit Ratios

4.4.5 Experiment Results

4.4.5.1 Cache Hit Ratios

The most important results of our experiments is depicted in figure 4.10. The curves correspond to the cache hit ratios as a function of the individual node cache sizes. For illustration purposes, we chose a database containing 1,500 pages pertaining to the central one-square-kilometer region. Each node's cache memory can hold between 15 and 150 pages (1% to 10% of the total region's pages) in increments of 15 pages (1%). Our shared cache approach shows significantly better performance in both its flavors (with and without data jettisoning) than the local caches acting alone. In fact, our approach yields at least a full tenfold performance improvement for the same amount of cache memory. Interestingly, the performance improvement tapers off around the 10% mark, indicating that further increasing the amount

of cache memory yields reduced benefits after that point.

It is important to note that this performance increase is a product not only of the individual node's cache memory, but also of the total number of nodes in the region. Recall that our analytical model indicates that in this scenario we expect to see between 20 and 30 nodes in the region at any point in time. Clearly, we would have to increase the cache memory as the node density decreases if we were to achieve the same performance.

4.4.5.2 Number of Cached Copies

We were not only interested in hit-ratio performance, but also in survivability. This was particularly important for military scenarios. The main measure of performance in this respect is the number of replicas of a given item among the nodes in the region. Though a higher value of this measure would obviously degrade the hit-ratio performance, it would also mean that a larger portion of the nodes can be suddenly lost with relatively minor effects in terms of overall performance degradation. Figure 4.11 shows the results of our analysis of the number of replicas in the region as a function of the size of the individual node's cache memory. When using our shared cache approach with data jettisoning, our approach is only slightly better in terms of number of replicas than a traditional local cache approach. However, when we consider the previous hit-ratio performance results, this small difference becomes significant because it underscores the effectiveness of shared spatial caching.

4.4.5.3 Popularity of Cached Copies

It is not enough to know the number of copies of each item in the shared cache. We also need to know that we are caching the *right* items. Since we are specifically interested in caching the items that are requested most frequently, their popularity is an important parameter. Figure 4.12 describes the relationship between each item's popularity (measured in total number of times it was requested), versus the number of copies of that item in the shared

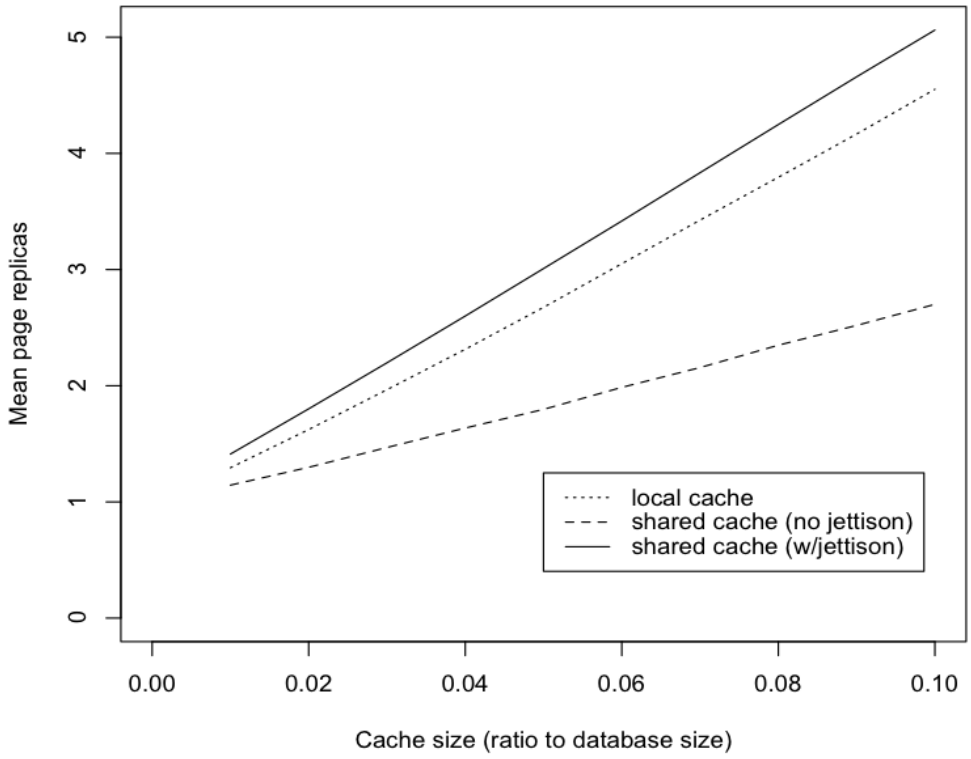


Figure 4.11. Number of copies of items in cache

cache. Note that the scales are orders of magnitude different, so we multiply the number of copies in the cache by a factor of 150 so that the shape of the distribution is discernible. The point of the graph is to show that, regardless of shared cache size, it tends to store more copies of the more popular (i.e., importnant) items. This means that the risk of losing local availability to that data in the event of nodes being destroyed is reduced.

4.4.5.4 Utilization of Each Node's Cache

Our final measure of performance is the degree to which each individual node's cache is being utilized in our framework. Clearly, it is best if every node's cache is fully populated by the time that node leaves the region. In every one of our experiments, all departing nodes fully utilized their available cache space. This means that our approach allowed even the brief visitors to completely load their caches and contribute maximally to the shared cache.

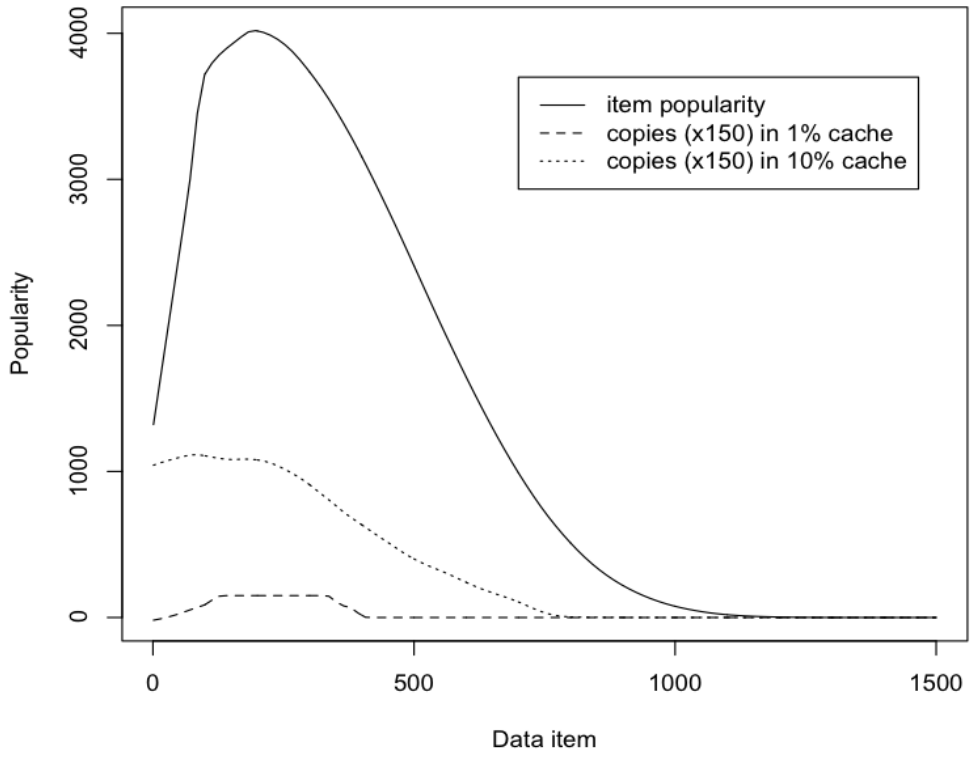


Figure 4.12. Popularity in database compared to copies in shared cache

4.5 Conclusions

By every measure of performance, our shared spatial cache is very successful. The results of both our analysis and simulations have shown that in mobile environments it works significantly better than local caching alone. In fact our shared spatial caches perform one order of magnitude better than local caches in terms of cache hit ratios. Furthermore, by carefully balancing performance and redundancy, the shared cache can tolerate the sudden loss of a significant number of its constituent nodes without a dramatic drop in hit ratios. This makes it uniquely suited to the emergency response and military scenarios for which it was created.

A critical open question that remains is the following. Under what conditions will this shared cache work. We have already stressed our idea that there must be a strong

correlation between the performance of this cache and the rates or arrival and departure from the region. Our work thus far, however, has simply depended on building a credible mobility scenario and seing how the cache works in it. We need to understand the full range of conditions under which our approach remains feasible. Doing so requires a much more rigorous approach, which is the subject of the next chapter.

CHAPTER 5

A Markov Chain Model of Shared Local Knowledge

5.1 Introduction

This chapter describes our model for a shared spatial cache as envisioned within the Ancile system. The idea is that local node exchanges can reduce or even eliminate the need for costly messages to and from a central server in a mobile ad-hoc network of peer-to-peer spatial databases.

5.2 The Need for a Markov Chain Model

Our work thus far, while sound and promising, lacks the mathematical rigour that would be expected of a doctoral dissertation. Specifically, we have not yet proved that *any* amount of information could persist within a spatial region by simply being handed off from node to node as knowledgeable nodes depart and ignorant ones arrive. Even the most fervent supporter of our work would be forced to ask the question: *when would this cache work and when wouldn't it?*

What we need, then, is a mathematically sound model of the interactions among the

nodes leading to information persistence. What we really care about is the number of copies of each fact that are present within the individual caches of the nodes in the region at any point in time. This is to say that, given a particular node and knowledge configuration, how can that configuration change from one instant to the next? If we substitute the word *configuration* with the word *state*, we have just described a Markov Chain process [52].

Markov Chain random processes have a number of attractive features with regard to our work. First and foremost, they exhibit what is known as the Markov property, which states that given a current state, the previous state (or any before even that one) is irrelevant in determining the future state(s). This property is attractive when modelling complex phenomena, because it is not necessary to track every state of every entity over time. Instead, we need only look at the current state and at the set of possible future states.

Another attractive feature of Markov Chain processes is that they lend themselves to statistical analysis. In other words, given a sound description of a Markov Chain process, a simple application of algebra reveals whether or not the process tends to spend a fixed part of its lifetime in a given set of states. This is particularly useful to us, because it can tell us whether or not our shared cache framework is stable in the long run and, if so, whether it tends to stay in states wherein the required information is contained within the shared cache. In other words, we can show mathematically that the system will probably remember a given number of facts under specific conditions.

5.3 A Simple Motivating Example

Suppose that our world consists of just two actors who are free to move around at will and learn and remember one (and only one) fact while they are inside of a specified region of interest. Once the mobile actors leave the region, they immediately forget the fact. The actors can only communicate when they are inside the specified region, which means that both actors must be inside that region if the fact is to be communicated. It is important for

Table 5.1. Motivating Example Markov Chain States

State	0	1	2	3	4	5
Nodes	0	1	1	2	2	2
Facts	0	0	1	0	1	2

Table 5.2. Events and their Transitions in a Simple MC

	0	1	2	3	4	5
0	N	A	0	0	0	0
1	D	0	G	A	0	0
2	D	L	N	0	A	0
3	0	D	0	0	G	0
4	0	D	D	L	0	G
5	0	0	D	0	L	N

our mobile actors to learn the fact as soon as possible, even if they later forget it. For this reason, each actor will try to get the fact from another actor as soon as it enters the region of interest.

We are interested in understanding how many actors and how many copies of the fact (either zero or one per actor) exist within the region of interest. We can encode this information in six distinct states as depicted in the state matrix S below, where the first row counts the number of actors inside the region, and the second row counts the number of copies of the fact inside the region. The columns could then be labeled zero through five to designate the six possible states.

We define the following possible events, which is to say state transitions:

A : actor arrival

D : actor departure

L : loss of information not due to a node departure

N : nothing happens

G : an actor gains (learns) a fact

Table 5.2 captures all possible transitions on the state space for our simple Markov chain. We can see from this, that the states in which not all nodes know the fact (e.g., 1, 3, and 4) are inherently unstable in that the probability of no state transition from them is exactly zero. This is because, in the absence of arrivals, departures, or information losses, nodes in these states are guaranteed to seek out the fact. Stable states are those wherein every actor in the region knows every fact (e.g., states 2 or 5 in our simple model).

This matrix is misleading, because for instance **N** events are not equally likely in different states. Furthermore, the probability of transitioning from state 4 (i.e., 2 nodes, 1 fact) to state 2 (1 node, 1 fact) requires that a node departs *and* that the departing node not be the one that already knows the fact. If the knowledgeable node were the one to leave, then the transition would be from state 4 (i.e., 2 nodes, 1 fact) to state 1 (i.e., 1 node, 0 facts). We can determine the probabilities of each event as follows.

P[A] : Probability of one arrival during the time unit. α

P[D] : Probability of one departure during the time unit. δ

P[L] : Probability of an actor spontaneously forgetting the fact. γ

P[G] : One minus the sum of the all other probabilities for the next state transition for unstable states

P[N] : One minus the sum of the all other probabilities for the next state transition for stable states

With these considerations in mind, we derive the transition probability matrix T below.

$$T = \begin{bmatrix} 1-\alpha & \alpha & 0 & 0 & 0 & 0 \\ \delta & 0 & 1-\alpha-\delta & \alpha & 0 & 0 \\ \delta & \gamma & 1-\alpha-\delta-\gamma & 0 & \alpha & 0 \\ 0 & \delta & 0 & 0 & 1-\delta & 0 \\ 0 & \frac{1}{2}\delta & \frac{1}{2}\delta & \gamma & 0 & 1-\delta-\gamma \\ 0 & 0 & \delta & 0 & \gamma & 1-\delta-\gamma \end{bmatrix} \tag{5.1}$$

To solidify our example a bit, we can assume $\alpha = \delta = 0.4$ (e.g., a mobile actor arrives and departs on average every other time unit). We also fix the value $\gamma = 0.1$, so actors spontaneously forget the fact with probability $U[0.1]$.Then, our transition probability matrix would be as is shown below.

$$T = \begin{bmatrix} 0.6 & 0.4 & 0 & 0 & 0 & 0 \\ 0.4 & 0 & 0.2 & 0.4 & 0 & 0 \\ 0.4 & 0.1 & 0.1 & 0 & 0.4 & 0 \\ 0 & 0.4 & 0 & 0 & 0.6 & 0 \\ 0 & 0.2 & 0.2 & 0.1 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0.1 & 0.4 \end{bmatrix} \tag{5.2}$$

We are interested in finding the unique stationary distribution π of this Markov chain (if it exists). To start off, we assume it exists. Therefore, we know the following equations are true.

$$0.6\pi(0) + 0.4\pi(1) + 0.4\pi(2) = \pi(0) \tag{5.3}$$

$$0.4\pi(0) + 0.1\pi(2) + 0.4\pi(3) + 0.2\pi(4) = \pi(1) \tag{5.4}$$

$$0.2\pi(1) + 0.1\pi(2) + 0.2\pi(4) + 0.5\pi(5) = \pi(2) \tag{5.5}$$

$$0.4\pi(1) + 0.1\pi(4) = \pi(3) \tag{5.6}$$

$$0.4\pi(2) + 0.6\pi(3) + 0.1\pi(5) = \pi(4) \tag{5.7}$$

$$0.5\pi(4) + 0.4\pi(5) = \pi(5) \tag{5.8}$$

$$\pi(0) + \pi(1) + \pi(2) + \pi(3) + \pi(4) + \pi(5) = 1 \tag{5.9}$$

Each of the preceding equations describe the probability of the system being in a given state. For instance equation 5.3 tells us that the probability of being in state 0 is the sum of the products of the probabilities of being in any other state and transitioning to state zero. We have a system of seven equations on six unknowns, so we can use some simple algebra to solve for π as follows.

$$\pi = \begin{pmatrix} 0.3416 & 0.2123 & 0.1293 & 0.0969 & 0.1198 & 0.0998 \end{pmatrix} \tag{5.10}$$

Since all elements of π are non-negative, then π is the unique stationary distribution for our Markov chain. In our very simple example, the net result is that we can expect that:

- the region will be empty (i.e., no nodes inside it) about 34% of the time,
- at least one node will be in the region and know the fact about 35% of the time, and
- one or more nodes will be in the region, but not know the fact about 31% of the time.

To validate our analysis, we ran a simulation of our Markov chain by choosing a random starting state and following the chain for 100 transitions. We ran this experiment 10,000 times and produced the results listed below, which match our predictions very closely.

Table 5.3. State Space Growth

Actors	Facts	States
1	1	3
2	2	14
3	3	100
4	4	979
5	5	12,201
6	6	184,820
7	7	3,297,456

$$\pi\prime = \begin{pmatrix} 0.3383 & 0.2097 & 0.1381 & 0.0928 & 0.1221 & 0.0990 \end{pmatrix} \tag{5.11}$$

5.4 A More General Model

It becomes infeasible to use our earlier manual approach to describe realistic systems potentially comprising hundreds of actors and thousands of facts. In fact, it may very well be computationally infeasible for us to do any useful analysis on such large models because of the growth rate of the Markov chains as a function of the number of actors (m) and the number of facts (n). Table 5.3 gives the reader an idea of just how fast this growth is.

What we need is general model of this system that is amenable to analysis regardless of the value of the parameters. To do this, we describe the state of the system using a pair $s_t = (x_t, y_t)$ where x is a discrete random variable with values in the range $[0..m]$ indicating the number of nodes in the region, and y_t is a discrete random vector of length n, where each value $y_t[i]$ is the number of copies of fact i in the region (i.e. known by actors in the region). Clearly, $1 \leq i \leq n$ (each fact has to be accounted for) and $0 \leq y_t[i] \leq x_t$ (there can be no less than zero copies of a fact nor more than the number of nodes in the region at the time).

5.4.1 Partial Order of States

In order to ensure a consistent sequencing of all possible states, we need a partial order of the state space S. To facilitate this, we assume that some facts are more important than others. How, precisely, one assigns these priorities is not relevant to our model. Given this assumption, we designate $y[1]$ to be the most important fact and $y[n]$ the least important one. This assumption is the basis for the partial order of states illustrated by the following example, in which $n = 3$.

$$(0,(0,0,0)) < ... < (2,(2,0,1)) < (2,(2,0,2)) < (2,(2,1,0)) < ... < (3,(3,3,3)) \tag{5.12}$$

In the example above, each state is defined by a tuple consisting of a scalar indicating the number of nodes in the region, and an n-tuple that indicates the number of copies of each of the n facts within the region. This means that the first state in the partial order exemplified above consists of zero nodes and zero copies of each of the three facts. The next state listed consists of 2 nodes, 2 copies of the first fact, no copies of the second fact, and one copy of the third fact. The partial order is established on the basis of a unique scalar value v assigned to each possible state. This value v is given by the following equation.

$$v = x_t^{n+1} + \sum_{i=1}^{n} y_t[i] x_t^{n-i} \tag{5.13}$$

In equation 5.13, x_t is the number of nodes in the region at time t. The value $y_t[i]$ is the number of copies of the ith fact and n is the number of distinct facts that are known about the region. In essence, this equation treats the state descriptor as an $n + 1$ digit number wherein each digit can take on values between 0 and x_t (the number of nodes in the region at time t).

5.4.2 Events

Transitions from each state in our model to every other possible state are triggered by events. Essentially, we capture the arrival and departure of actors from our region of interest. For each actor inside the region, that actor can learn a fact, or forget one. Finally, in some states it is possible for nothing to happen and therefore there is no transition. Each of these events is described in detail in the following subsections.

5.4.2.1 Actor Arrival

The first transition is that which takes place when an actor arrives in the region of interest. We previously stated that this happens with probability α, which has some (as of yet undefined) distribution. In this situation, the value of $x_{t+1} = x_t + 1$, provided that $x_t < m$. Otherwise, the transition has probability zero, since we can't have more actors in the region than the total number of actors. Therefore, this transition takes the form:

$$(x, y) \rightarrow (x + 1, y), x < m \tag{5.14}$$

Note that, though the value of y remains unchanged, the upper bound on the value of its elements increases by one, since there is one more actor in the region that can learn facts. Since y is fixed, the total number of possible transitions on an arrival event is exactly one.

5.4.2.2 Actor Departure

The second transition is one that occurs when an actor leaves the region of interest, which, as before, happens with probability δ. This situation is slightly more complicated, because the departing node may have known one or more facts. If our model kept track of which facts were known by which node at all times, then this transition would be very straightforward.

However, we choose to keep only aggregate statistics in order to keep the model manageable.

While calculating the value of x_{t+1} is a straight-forward subtraction of one (provided $x_t > 0$), the value of y_{t+1} is one of several possibilities. Every non-zero element of y_t may decrease by one (with equal probability $\frac{1}{x_t}$) because we assume every fact is equally likely to be known by the actor that departed. Furthermore, every element of y_t greater than x_{t+1} must decrease by one because the number of actors who know a given fact can never be greater than the total number of actors in the region.

$$(x, y) \rightarrow (x - 1, y'), x > 0 \tag{5.15}$$

$$\forall i, 0 < i < n \text{ let } y_{t+1}[i] = \begin{cases} 0 & \text{if } y_t[i] = 0 \\ y_t[i] - 1 & \text{if } y_t[i] > x_{t+1} \\ y_t[i] - 1 & \text{with probability } \frac{1}{x_t} \\ y_t[i] & \text{otherwise} \end{cases} \tag{5.16}$$

Calculating the total number of possible transitions on a departure event is more complicated, so we should spend a bit more time thinking about it. When the transition is to state $X = 0$ there are no actors left in the region so there can only be one state, namely, the one wherein $Y = (0, 0, ...)$ since there is no one in the region to know any of the facts. When the new $X = 1$, there is exactly one actor in the region, so the possible states are for that actor to know none, some, or all the n facts. Accordingly, there are 2^n possible values for the Y vector. When there are two actors in the region, then each fact may be known by none, one or both actors, so the total state space is 3^n possible values for the Y vector. In general, then, for any given value $X = x_{t+1}$ there are $(x_{t+1} + 1)^n$ possible states. This means that we can group all possible states according to the value of X, as shown in figure 5.1.

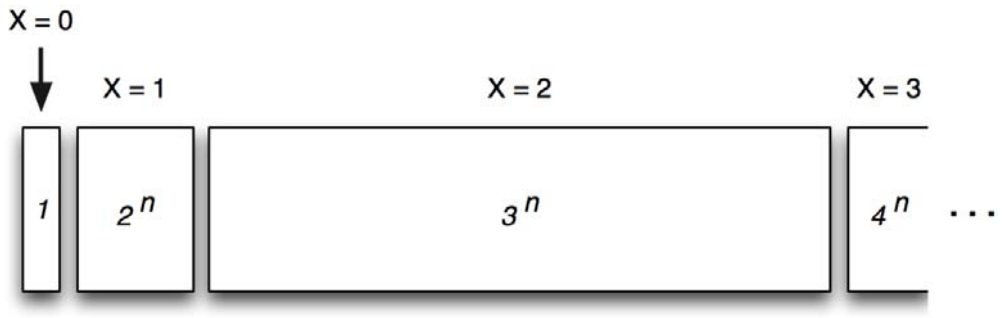


Figure 5.1. Possible number of states as a function of X

5.4.2.3 Information Loss

Though it is not desirable for nodes to loose information, our model accounts for this situation. Possible real-world causes of information loss include information items that take up more memory than others, information that is no longer interesting to the actor, and information that somehow expires. We can describe the lost information using a vector y' of length n that consists of zeros in all places but one, say the jth position. This indicates that one copy of the jth item is lost by some actor in the region.

$$(x, y) \rightarrow (x, y - y') \tag{5.17}$$

Information loss can happen in one of various ways. First, an actor may simply decide to discard a random information item, which means that there would be n possible next states on an information loss event. Secondly, the actor may discard the fact with the lowest value, in which case there can only be one subsequent state. Finally, the actor may attempt to discard the fact that is known by the most number of actors in the region. In this final case, it would be possible for there to be one or more (up to n) states depending on how many facts are known by the same number of actors. For the purpose of our initial model, we will assume that nodes discard the fact with the lowest priority, though this policy will change later on.

5.4.2.4 Information Dissemination

Another possible state transition (or more accurately, *set* of state transitions) takes place in the absence of actor arrivals or departures. This is where actors learn facts. As before, we use a vector y', but in this case the single non-zero value in it indicates the fact that is learned by some actor in the region.

$$(x, y) \rightarrow (x, y + y') \tag{5.18}$$

In this transition, let j be the index of the new fact that an actor has learned. In keeping with the idea of prioritized facts and the partial order they induce (discussed in section 5.4.1), we use the convention that information dissemination attempts to maximize the value of the single fact that can be acquired in one event. Therefore, for any given state, there is only one next state and that is where the most significant element of Y that is not yet equal to X is incremented. Note, however, that we can use any of the three policies mentioned in section 5.4.2.3.

5.4.2.5 Nothing Happens

This final state transition is characterized by no changes to either x or y. This is only possible when all actors (if any) in the region already know all the existing facts.

$$(x, y) \rightarrow (x, y) \tag{5.19}$$

We assume that this event is very rare, because actors want to acquire information. In the absence of arrivals, departures, or information losses, the actors in the region will attempt to learn a new fact rather than do nothing. Obviously, if all actors in the region already know all the facts, then it would be possible to remain in the same state until the next actor arrival, departure, or information loss.

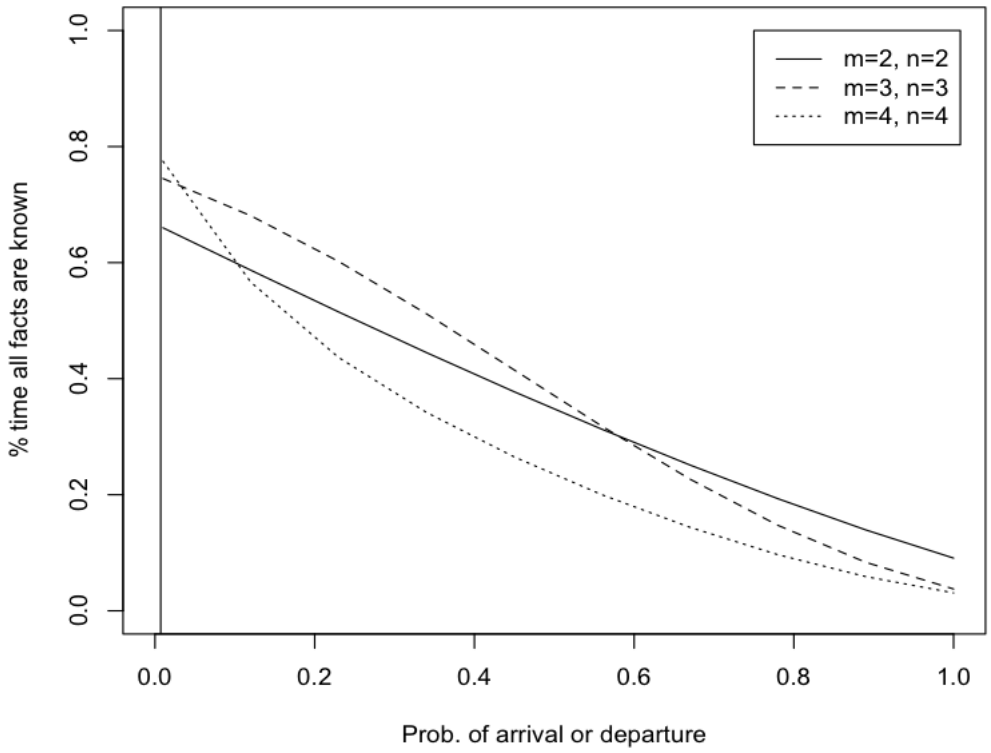


Figure 5.2. Percent of Information Coverage

5.4.3 Analytical Results

Using the algorithms described in section **??**, we can build transition matrices for arbitrary values of our parameters. In practice, however, these matrices become too large when $m+n > 10$. We can, however, use them to solve systems of equations, and hence determine standing distributions, for scenarios with very small m. Figure 5.2 presents the results of solving the systems of equations given by the transition matrices for three cases of (m, n) with values $(2, 2), (3, 3)$ and $(4, 4)$. Though these are admittedly trivial cases in the real world, they serve as a springboard upon which we build our analysis of this system.

The horizontal axis of figure 5.2 shows the probability of an arrival or departure event $\alpha + \delta$, while the vertical axis shows the percentage of the time that the Markov chain model will spend in a state within which every fact is known by at least one node. Recall that we assume the model to be in a steady state, which implies that $\alpha = \delta$. The three curves

depicted capture the data for each of the scenarios we considered.

It should not be surprising that, as arrivals and departures become less frequent (i.e., the actors who are in the region tend to stay in the region), the amount of facts known by them increases as well. Recall that we are not interested in maximizing the number of actors in the region, we just want to know the conditions under which they will be better informed.

The vertical line near the origin is the point where α and δ take on the values for our queue theory model for 30 nodes in a 9 square kilometer world with a 1 square kilometer region of interest in its center. (We refer to this scenario as the random30 model.) This serves to illustrate that we expect the values of these probabilities to be very small in the real world. It also shows that our model shows a great deal of promise for realistic applications.

5.4.4 Stochastic Simulations

We know that we cannot solve the systems of equations given by our Markov chain model for the cases we considered in our queue theory models earlier. As we stated earlier, we are particularly interested in what we call the random30 model because it mimics an infantry platoon or search and rescue effort of reasonable size operating in a fairly large area. We are also interested in exploring the situation wherein the communications links among the actors have a bandwidth of one megabit per second. Given these parameters, we want to run our Markov chain model with $n = 7$ (the mean number of actors in the region is 6.72) and $n = 5$ to represent that number (in hundreds) of pages in an actor's cache. In this specific case, interarrival times are random variables with a Poisson distribution and parameter $\lambda = 0.007149$. This means that the expected value for $\alpha = 0.0055135$.

A challenge in running these simulations is the sheer size of the transition matrices. The transition matrix for the case $m = 7, n = 5$ consists of $61,776^2$ cells, which requires more than 15 gigabytes of memory in our computer's memory! Fortunately, the matrices are sparse, and Roger Koenker and Pin Ng developed a software package that allows R called

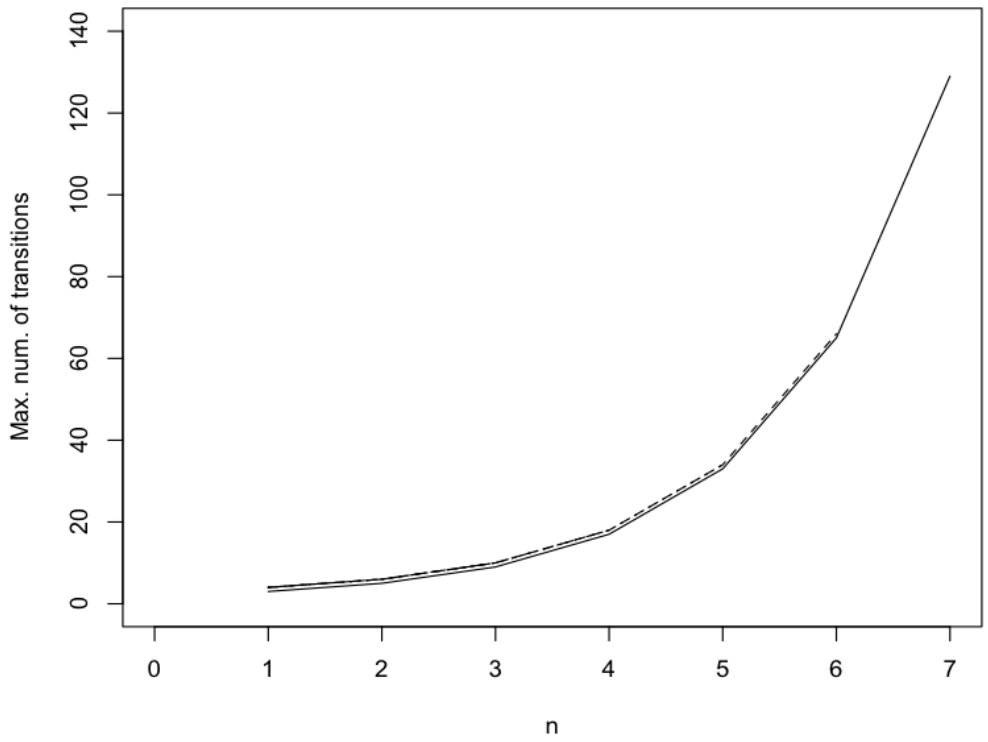


Figure 5.3. Growth in the max. number of state transitions

sparsem [28] to use sparse matrices. Still, the transition matrix for the 7/5 case was large enough that it required the use of 62 files each of which store just 1,000 rows of the matrix. Generating this matrix required about ten hours of computations.

Figure 5.3 shows the growth of the maximum number of transitions from a given state as a function of n. Note that this is the number of *actual* possible transitions, which is the same as the number of non-zero entries in the transition matrix. Though it is hard to tell, that graph actually plots the curves for m in the range [2..6], which is all the scenarios that we can manipulate on a computer with 4 gigabytes of memory. All plots coincide perfectly for n greater than 2. We conclude that the number of transitions t is given by $t = 2^n + 2, n > 2$. This means that in our case of interest, we would only need $61,776 \times 34 = 1,050,192$ total entries in our matrix, which requires approximately 8.4 megabytes of memory. This is almost two thousand times less space, and allows us to simulate the chain for the scenario of interest.

Our stochastic simulations consist of 1000 experiments. We consider two cases: one

Table 5.4. Stochastic Simulation Results

	No nodes in Region	Nodes, no Facts	Nodes, Some Facts	Full Fact Cover
Random Start	0	0	0.333	0.667
Full Cover Start	0	0	0.217	0.783

in which the start state of the Markov Chain is (uniformly) randomly selected among all states, and another in which the start state is randomly selected among states in which each fact is already known to at least one node. This second case is important in understanding the relative permanence of information within the region in the absence of communications to a central data source outside the region.

Each experiment starts at a random state and follows the chain for 100 transitions, recording the last state. Each transition is determined by generating a uniformly distributed random number in the interval [0..1]. The probabilities of each transition out of the present state are considered sequentially in the order in which they appear in the transition matrix. The transition whose probability brackets the random number is then selected. For instance, if the allowed transitions out of a given state are on an arrival with probability 0.3, on a departure, with probability 0.3, and on information sharing with probability 0.6, and the random number generated is 0.5, then the corresponding event would be a departure.

Table 5.4 summarizes the results of our stochastic simulations. For brevity, we only include broad classes of states, rather than the specific states themselves. We are not interested in knowing how many copies of a given fact exist within the region, just whether or not someone knows that fact at the end of the simulation.

The fact that we always end up in a state wherein there are nodes within the region with *some* knowledge is consistent with our earlier analysis. According to our queue theory model, we did not expect to ever see an empty region. Therefore, we did not expect complete information loss either.

Similarly, the probability of each fact being known to at least one node in the region

is consistent with our analytical model, which we discussed in section 5.4.3 above. We conclude that our models (queue theory, Markov chain analysis, and Markov chain stochastic simulation) are consistent with each other.

5.5 Discrete Event Simulations

Having reached the end of the road in terms of what we can analytically model, it is time to turn our attention to the development of models based on our earlier work, but that lend themselves to discrete event simulation. Though this simulation technique is not exhaustive or mathematically rigorous, it does allow us to explore scenarios of significantly greater complexity and, hence, realism.

5.5.1 Simulation Environment

We use the discrete event simulator ns-2, which is widely used in academia and industry for network simulations. It is open-source and supported by a large community of developers, which means that it has organic support for a wide variety of networking technologies. It is also built in such a way as to facilitate the development by third parties of new or experimental protocols and technologies.

ns-2 does not fully support simulations at the application-layer. Fortunately, a development team out of the Naval Research Laboratory developed a simulation engine called AgentJ. AgentJ, essentially, is an interface between Java classes and ns-2. It allows us to develop a system in Java just as we would for use in a real network, and then run this system on a simulated ns-2 network, taking full advantage of this simulation engine's rich feature-set.

5.5.2 The Java Knowledge Tracker

At the heart of our first set of simulations is a Java class that manages a global matrix of actors and facts within the region. This class is called KnowledgeTracker and is depicted in figure 5.4. Quite simply, it keeps track of who knows what when. In accordance with the AgentJ application programming interface (API) definitions, this class exposes a public method called `command()`. It is through this method that ns-2 interacts with the class by sending it commands to execute. These commands are implemented in the various private methods that are responsible for adding and removing nodes from the knowledge matrix as well as for exchanging information among the nodes based on the mode of the manager.

The KnowledgeTracker class can be explicitly configured to operate in one of three modes. The default mode is PRIORITY, which ensures that all nodes possess the highest priority fact first, and then facts with lesser priorities are sequentially shared. The RANDOM mode, on the other hand, randomly shares facts among the nodes in the region without regard for their relative importance. Finally, the BREADTH mode attempts to ensure that *some* node knows each fact within the region and then uniformly and progressively increases the number of copies of each fact.

It is also possible for the KnowledgeTracker class to call upon ns-2 functionality. To do so, the class includes a reference to an AgentJ object, which exposes functions such as determining the ns-2 address of a network node, getting the current simulation time, and others.

There are also two supporting classes named NodeCache and Fact that model the collection of information present in a given node. These are container classes and as such consist of encapsulated data values with the appropriate getters and setters. In addition, the NodeCache class has a method `contains()` which determines whether a given fact is part of a particular cache. Also noteworthy is the `equals()` method of the Fact class, which is useful to test whether to instances actually refer to the same fact.

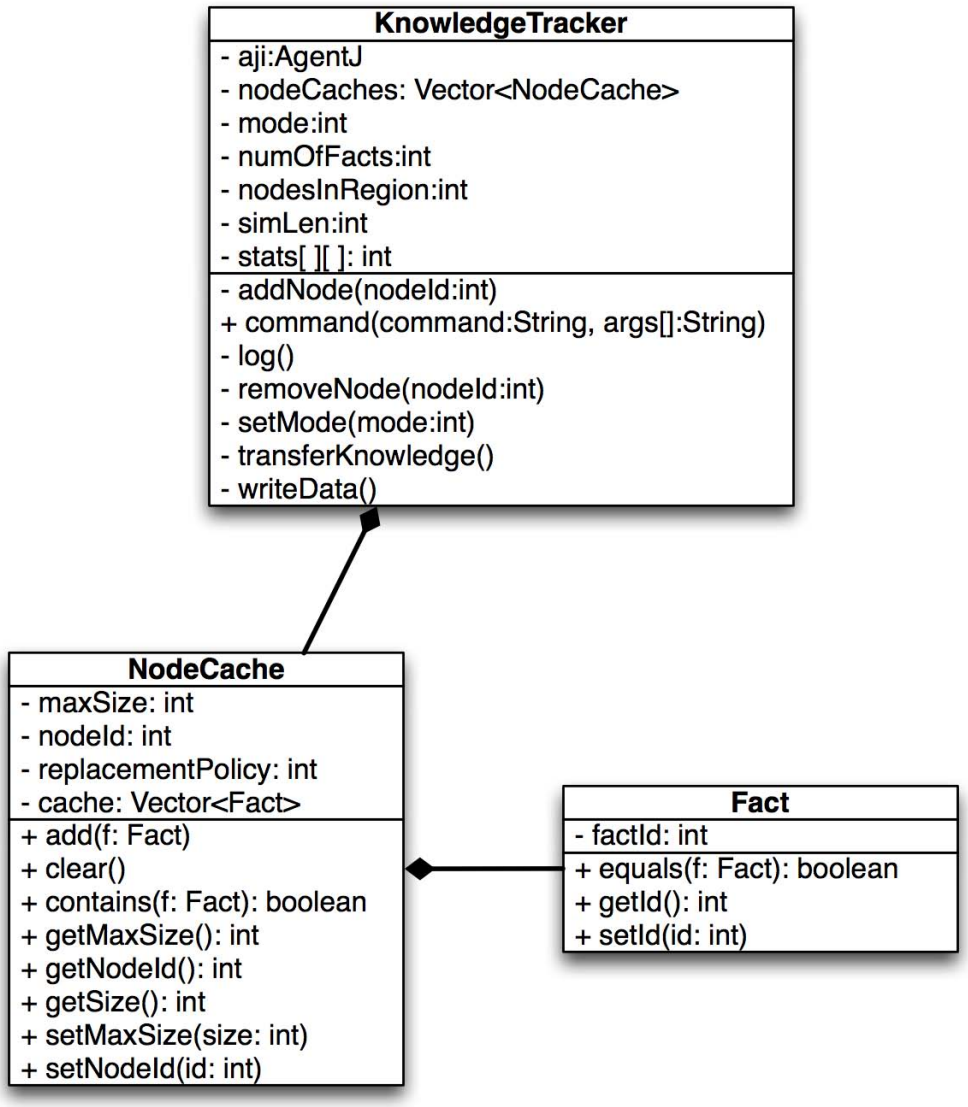


Figure 5.4. UML Diagram for the Java Classes

5.5.3 Building Activity Scripts

Since the Knowledge Manager has no knowledge of individual node activities, it must be told exactly what to do at each simulated time interval. To this end, we use ns-2 scripts that call on the `command()` interface of the class to execute the appropriate actions.

First, we use the BonnMotion mobility scenario generation and analysis tool to create 30 mobility scenarios. Each scenario encompasses a square geographic area measuring 3000 meters on each side. Within this area, 30 nodes randomly move about using the Random

Waypoint mobility model, which is widely used in mobile ad-hoc network research. This is the same process we followed to validate our queue theory model earlier, but we use 30 new scenarios.

Once the 30 scenarios were built, we used a script to analyze the motion of the nodes and determine when each entered or departed the central one-square-kilometer region of interest. When a node entered the region, the script generated a command to add the node to the KnowledgeManager matrix. Similarly, a command was generated to remove nodes as they left the region of interest. Finally, if neither arrivals nor departures occurred in a given second interval, the script generated a command to transfer knowledge among the nodes present in the region. The commands are recorded in 30 script files that ns-2 can import and use. Code listing 5.1 illustrates some the of the commands in an ns-2 script that are passed to the Java class KnowledgeManager.

Listing 5.1. Sample ns-2 Script

```
$ns_ at 3.0 "$km agentj add-node [$node_(7) node-addr]"
$ns_ at 5.0 "$km agentj remove-node [$node_(1) node-addr]"
$ns_ at 1.0 "$km agentj transfer-knowledge"
```

5.5.4 Experiment Setup

Having generated 30 ns-2 scripts based on the analysis of the mobility scenarios, we sequentially run each script through ns-2 with the AgentJ plugins varying the number of total facts in the region from 100 to 1000 in increments of 100. Each simulation lasts 3600 seconds and involves the 30 nodes moving through the nine square kilometer simulation area, exchanging information. At the beginning of each experiment, the first node to be added to the region is given every known fact. All other nodes have to acquire copies of each fact from this first node before either leaves the region. No external sources of information are allowed. If the only node to know a fact leaves the region before it is able to transfer it to another node,

then that fact is effectively lost for the remainder of the simulation. All nodes forget all facts they may have known immediately upon departing the region.

In time periods wherein an arrival or a departure occurs, no knowledge transfers can happen. Whenever there are no arrivals or departures, a single source agent provides a single fact to a single recipient. Obviously, a broadcast transmission would be much more effective, but we are looking for an approximation of a lower bound on the amount of information within the region and so are not optimizing the mechanisms. We assume that all agents within the region are directly connected to each other (i.e., there is no need for intermediaries in an information exchange). We also assume that a fact can be transmitted in a single time unit from source to destination.

At the end of each experiment, the KnowledgeTracker class is directed to write to a file the state of the system at each time interval. This gives us very detailed data about the state of the system at each time interval of each experiment. These 30 results files are then analyzed by an R script that determines the statistics of interest. We are specifically interested in knowing the fraction of the time during which every fact is known to at least one node in the region (total information coverage).

Additionally, we want to know what the minimum number of facts is that can be expected to persist within the region of interest. We hypothesize that this value is somehow dependent on the stability of the region as measure by the parameters λ and μ. To this end, we run a second set of experiments in which we randomly assign values to these parameters in the interval $(0, 0.1]$ with the constraint that $\lambda > 5\mu$. The factor five was chosen heuristically to minimize the probability that the region would be free of agents at any time. We generate 30 sets of values and then use these to generate arrivals and departures based on a Poison and exponential distribution (respectively). We also use the constraint that the region must never be empty (i.e., at least one node in the region at all times). These artificial mobility scripts are then run through ns-2 in order to determine how many facts survive each of the 30 simulations.

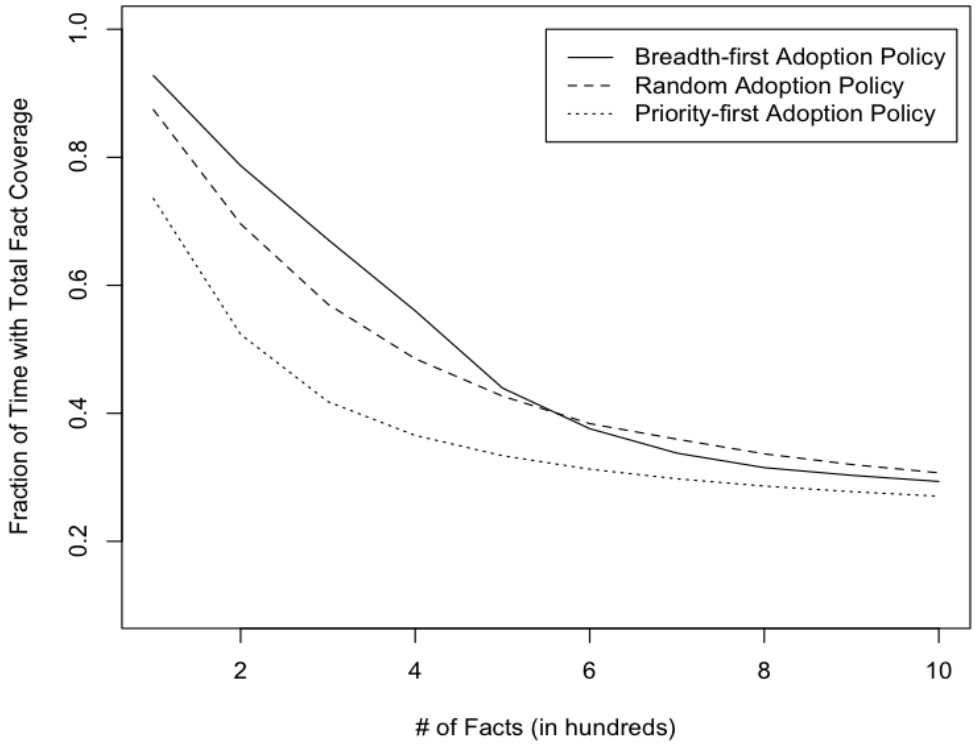


Figure 5.5. Probability of Full Cover

5.5.5 Simulation Results

We are interested in the fraction of the time that the system spends in a state wherein each fact is known to at least one agent. When this is the case, agents in the region of interest need not search for information outside of it, but rather are able to satisfy their requirements using facts that are already present in a neighbor's cache. We call this total information coverage. Figure 5.5 shows the percent of the time that our nodes experienced total information coverage during the simulations. We show this value as a function of the total number of facts (in hundreds) pertaining to the region.

Unsurprisingly, the fraction of the time spent in total information coverage drops exponentially as the number of facts increases. Among the three policies we use for information sharing, the breadth-first approach performs at least as well as the priority-based approach and they both outperformed the random approach.

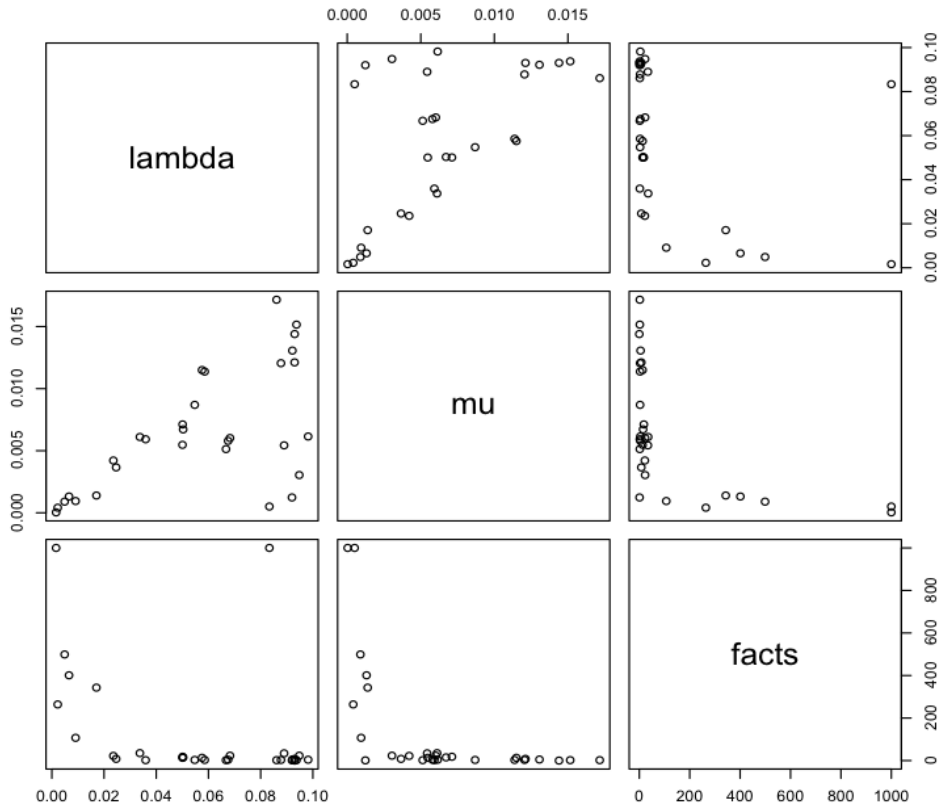


Figure 5.6. Correlation between lambda, mu, and the number of persistent facts

With regard to the second set of experiments aimed at determining the effects of the arrival and service rates on the number of permanent facts, table 5.5 and figure 5.6 summarize our findings. In the graph, we do a pairwise comparison of the three values (λ, μ, and facts). The sub-figures of interest are the top-right graph showing facts vs. λ, and the middle-right one showing facts vs. μ. It is clear that there is a strong non-linear correlation in both of these cases. This suggests that our hypothesis that the two parameters determine the number of permanent facts is likely to be true.

When plotted on a log-log graph in Figure 5.7, a fitted line drawn through our data points is almost linear, particularly with respect to λ. The line shown was plotted using the locally-weighted scatterplot smoothing (LOWESS) local regression method developed by William Cleveland [9]. It is important to note that we remove from the result data set a case in which zero facts were retained, since its inclusion would have made it impossible to

Table 5.5. Discrete Event Simulation 2 Results

λ	μ	Num. of Facts
0.0170767958508804	0.00139122942928225	343
0.0667275022948161	0.00513125550933182	2
0.092160612065345	0.0130641677184030	5
0.0246365285012871	0.00365742098074406	8
0.0065772928064689	0.00131318417843431	401
0.0860961419763044	0.0171605781884864	2
0.0833466920768842	0.00050390821415931	1000
0.0503094948362559	0.00671654243487865	15
0.035939112608321	0.00592244444414973	2
0.00909755218308419	0.000945718982256949	107
0.0930345270084217	0.0143947238335386	0
0.00162670314311981	3.32935713231564e-05	1000
0.0930064950371161	0.0121145320823416	8
0.0547386124264449	0.00869238164741546	3
0.0500492826104164	0.00547171765938401	13
0.0235702325124294	0.00421099185477942	22
0.0575390998041257	0.0114964447915554	13
0.0585704990895465	0.0113755072467029	3
0.0981863465858623	0.00614571017213166	4
0.0682568289572373	0.0060218554455787	23
0.0675321197137237	0.0057922100648284	4
0.0889826279366389	0.00543251363560557	34
0.0500901538413018	0.00711655190680176	18
0.0337271489901468	0.00610604907851666	35
0.0920020649209619	0.00123498667962849	1
0.0937294360715896	0.0151562129147351	2
0.0947916420409456	0.00304052776191384	23
0.00489421677775681	0.000898061762563884	499
0.0877897203201428	0.0120498465839773	3
0.00224875130224973	0.000409234105609357	264

use a log-log plot.

LOWESS is a non-parametric regression method, meaning that we do not need to specify the parameters for our regression function. It also means that we don't get coefficients for these parameters from the regression. However, since our interest ins in identifying the conditions in which our model performs well, we don't really need to derive a predictive formula. LOWESS is also a robust technique in that it does well in the presence of outliers, which were present in our data sets.

When we run a dedicated regression of the form $\log(facts) \sim \log(\lambda) + \log(\mu)$, LOWESS yields a model with a very low residual error or 1.284. The R script used to generate Fig. 5.8 is provided in listing A.2.

The graphical depiction of our regression is not surprising. We can logically expect the system to perform best whenever the arrivals are frequent and their dwell time is large. This situation is characterized by the point that is closest to the viewer in Fig. 5.8, which shows a large λ and small μ yielding a large number of persistent facts in the region. Conversely, the worst situation is when both λ and μ are large, because this represents the most chaotic situation within the region with many arrivals and departures happening constantly. Under these pathological conditions, there is little hope of transferring knowledge to an agent who will have enough time to pass it on to others before leaving the region. Finally, for small enough values of λ, μ appears to have a limited effect on the number of persistent facts within the region.

5.6 Conclusions

We have shown an estimated lower bound on the expected amount of information that can persist within a spatial region. This bound is determined by the rate of arrivals and the subsequent dwell time of these agents within the region. Clearly, this is a dynamic system, and we expect these parameters to vary over time. If we have fairly accurate estimates of

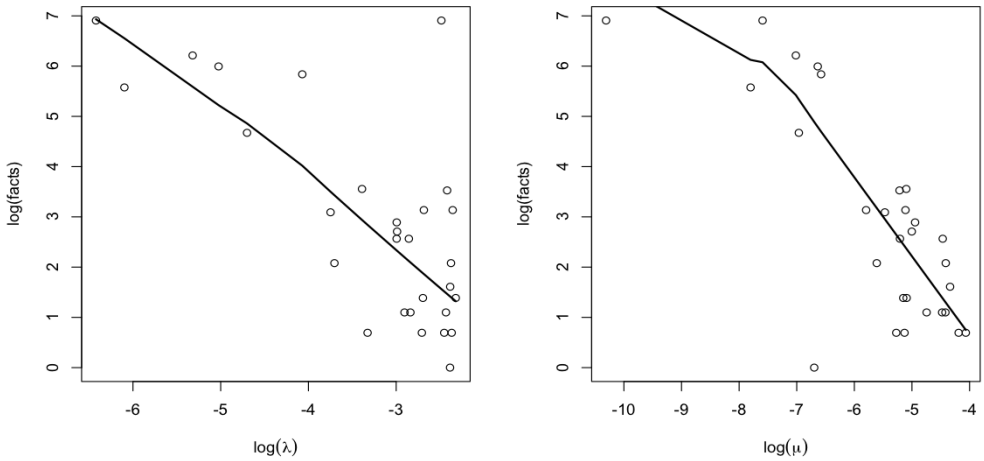


Figure 5.7. Log-Log plots of facts as a function of λ and μ

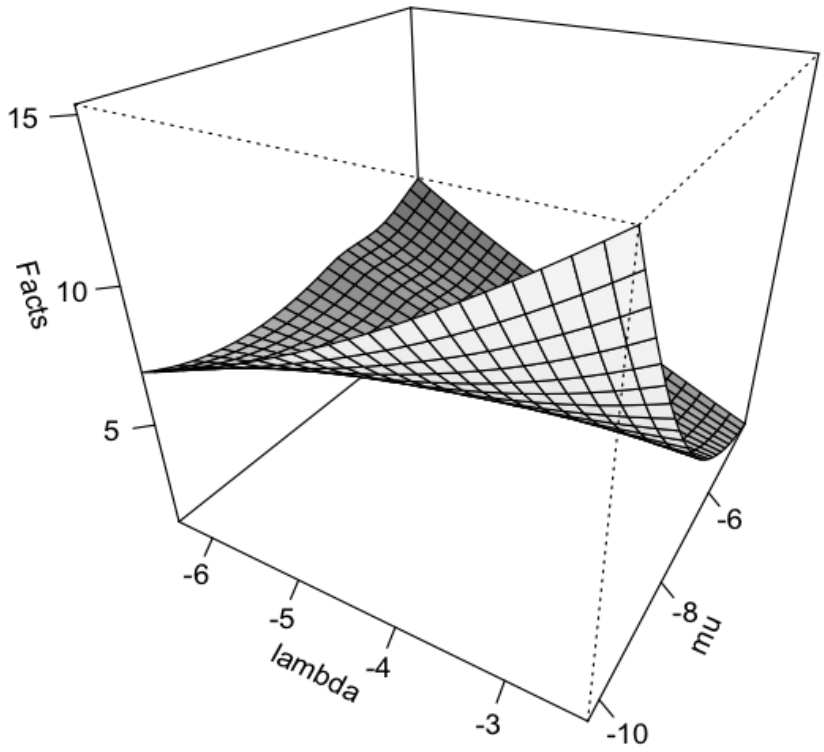


Figure 5.8. Nonparametric regression results

their values, however, we can predict the amount of information that can persist within the region while these values hold. Armed with this information, agents within the region can adopt information sharing strategies that ensure the persistence of critical information. We describe a mechanism for accomplishing this in the next chapter.

CHAPTER 6

Ethical Considerations

Most researchers are familiar with the requirements to consider ethical issues whenever human subjects are involved in their efforts. Though this dissertation has no human subjects it, like any report of scientific findings, has the potential to affect the lives of those who had no part in this research effort. Whether it is used by other researchers as a basis for their own investigations or it finds its way into the commercial sector wherein customers will depend on it, our research must and does stand up to the highest ethical standards. In this chapter, we address some important ethical concerns that are very relevant to the work at hand.

6.1 Computer Ethics

By most accounts, the need for a dedicated field of ethics in computing was first put forth by MIT professor Norbert Wiener in his book *The Human Use of Human Beings* [54] in 1950. Wiener was involved in the development of the first anti-aircraft weapon system with automated fire control. He realized that ethics must play an important role within his newly-created field of cybernetics and more broadly, in the use of computers. Unfortunately, his work in this endeavor failed to gain momentum until almost three decades later.

In the 1970s, Walter Maner was teaching medical ethics when he noticed that new ethical issues arose whenever computers where involved in medical ethics. This realization

spurred him to propose a new discipline of Computer Ethics. Maner then traveled widely, giving talks on Computer Ethics to diverse audiences. He also published his *Starter Kit on Teaching Computer Ethics* [33]. Maner inspired many in the field, including Deborah Johnson, who published the first text book on Computer Ethics [25] in 1998.

Definitions of the term Computer Ethics abound, but many practitioners refuse to tie themselves to one specific one. In the interest of clarity with regard to our own efforts, we define Computer Ethics as the philosophical study of the moral values and rules of the development, use, and study of computational systems.

6.2 Responsible Conduct of Research

Besides Computer Ethics, however, it is important to consider that this is a scientific investigation and, as such, it should adhere to established research ethics standards. An excellent set of standards in this regard is the one used by the National Institutes of Health [45], which is outlined in bullet form below and annotated to illustrate how it applies to this research effort.

Honesty Though, obviously, we do not in any way falsify, misrepresent or fabricate data, it is equally important to point out that we do not pick and choose the data that we *do* present in order to support conclusions that would otherwise be false.

Objectivity While the principle of honesty described above pertains to *intentionally* misleading the readers, it does not address the unintentional deceptions that result from our own biases. We believe our work is objective and make every effort to consider all possible interpretations of our data so that it remains so.

Carefulness We have, on occasion, discarded fairly large data sets because we could not be certain of their precision. Likewise, we spend considerable amounts of time proofreading our writings and verifying our data to ensure we purge them of errors.

Openness Our data, programs, and scripts are available to third parties so that our results

may be examined and replicated. In fact, we welcome such efforts, as they can only serve to validate our work.

Respect for Intellectual Property All contributions made by others to our work is properly documented, even when those contributions have been tangential.

6.3 Consequences of Research

Finally, we must consider the future consequences of our research. Should this framework ultimately be adopted by others, what ethical implications must be addressed? The main issues deal with security and privacy. Though others, no doubt, exist, it is these that we want to stress in this document.

6.3.1 Security

Because our research has a view towards supporting military and emergency response situations, the issue of security is paramount. Though this, in itself, would make a good separate research project, we want to mention some salient points. Perhaps these will serve as a springboard for future work.

United States Code [53] defines information security as protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide

- integrity, which means guarding against improper information modification or destruction, and includes ensuring information nonrepudiation and authenticity;
- confidentiality, which means preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information; and
- availability, which means ensuring timely and reliable access to and use of information.

In this context, we must address the issues of security in transit and security at rest. This means that we are concerned with the security of the data, not only when it is being

exchanged among agents, but also while it is in a storage medium. The logical solutions to these issues is the use of a Public Key Infrastructure (PKI) [1], albeit with adaptations to take into account the resource-constrained environment of mobile devices.

6.3.1.1 Integrity

The use of PKI digital signatures is the traditional approach to ensuring information integrity. If a message is properly signed thus by its sender, it is practically impossible to modify it without the recipient being able to detect the alteration. By securely storing this signature in the database, it is equally possible to detect alterations to an item while it is at rest.

6.3.1.2 Confidentiality

The traditional way to protect confidentiality is through the use of symmetric encryption. This type of encryption, unlike PKI's asymmetric version, has the advantage of requiring significantly less computing. It is also widely available in many commercial radio systems.

Because of the potential military applications of our work, it is also important to consider the case wherein a device is captured by adversarial forces. Confidentiality can be assured in this case through the provision of a self-destruct mechanism for the data and/or the device. Additionally, it is possible to remotely wipe the data on a device as commercial vendors have already demonstrated in consumer applications.

6.3.1.3 Availability

Clearly, our framework is built on the very notion of ensuring the availability of the information whether or not contact is possible with external servers. Therefore, the main threat in this regard is that of interfering with the exchange of information. This is normally done through jamming of the radio spectrum. The logical counter-measure to mitigate this threat

is the use of spread-spectrum radios that make jamming much more difficult and hence localize its effectiveness to a smaller area.

6.3.2 Privacy

Because our work is heavily geared towards military and emergency response scenarios, privacy issues are secondary because most personnel in those scenarios have no such expectation. However, should this framework be adapted to other applications, it is important to provide mechanisms by which the privacy of the users can be ensured.

Specifically, our use of beacon signals that transmit the identity and location of a user would have to be anonymized somehow. A possibility is the use of coded identifiers that are only decipherable by friends or system administrators. This idea would allow our system to work, albeit in a more limited fashion, since only friends would have access to key information.

CHAPTER 7

Conclusions

In this Chapter we provide a summary of the conclusions and contributions of this dissertation. In addition, we point to directions in future work.

7.1 Summary of Results

- *Chapter 1 Introduction*: The introduction to this dissertation included the motivation for this project and our problem statement.
- *Chapter 2 Literature Review*: In this Chapter we presented relevant work in the areas that form the basis of this dissertation, which include: geocasting, distributed, spatial, and mobile databases, and data caching.
- *Chapter 3 A Stochastic Approach to Geocasting*: Our approach to the problem of moving information from its sources to its destinations is presented in this chapter. This approach, based on probabilistic geocasting, was shown to work much better than any of the other approaches considered. This was particularly true of pathological network topologies, in which it was the only approach that was able to deliver the majority of the messages. Across the spectrum of scenarios, we were able to deliver messages better than the others, and at transmission costs that did not exceed theirs.
- *Chapter 4 A Cooperative Spatial-Aware Shared Cache*: In this chapter, we transition from moving information to the places wherein it is relevant, to keeping that informa-

tion in those regions. We propose a cooperative spatial-aware shared cache that allows nodes to pool their resources in such a way as to provide most of the information required by the nodes in the region with no need to contact the remote data servers. This approach yields remarkably good information hit-ratios and is very fault-tolerant.

- *Chapter 5 A Markov Chain Model of Shared Local Knowledge*: After developing our shared cache in the previous chapter, we re-look the issue with more mathematical rigour here in order to identify the range of conditions under which is performs well. Our analysis and experiments show that the shared cache works in a wide range of realistic scenarios and quantifies what these scenarios are.

7.2 Summary of Contributions

The main contributions presented in this dissertation can be summarized as follows.

- Development of a conceptual framework for pervasive situational awareness. Situation awareness (SA) is the perception of environmental elements in space and time that are relevant to a particular individual or group. Our work has already laid the groundwork for an environment within which SA is disseminated to those who need it. This contribution was the subject of a recent article in a special edition of IEEE Internet Computing magazine [35].
- Development of a stochastic mechanism for transporting information across a mobile network. This mechanism is particularly well suited for dealing with adverse network conditions such as limited battery power, high node mobility which results in frequent topology changes, the sudden loss or destruction of nodes, and the presence of obstacles that impede direct communications. Part of this work was published in the 2009 International Conference on Information Technology: New Generations [37].
- Development of a mechanism by which nodes can maximize both local and global knowledge preservation and diffusion with limited power and storage capabilities. This

allows the nodes to make room for new information by sensibly discarding knowledge only when it is likely that other nodes have chosen to preserve it, or when that piece of information is no longer relevant.

- An understanding of the conditions under which information is lost, exchanged, and preserved in a peer-to-peer mobile geospatial database system. This is an important contribution to the study of emergent knowledge processes (EKP), which are organizational activities that exhibit emergent processes of deliberation involving complex information and actors with unpredictable roles and prior knowledge [34]. Clearly, EKP is a topic of much relevance to our principal problem scope: emergency response and military scenarios.

7.3 Future Work

We conclude this dissertation with a series of task that can be carried out as future work to complement this work.

7.3.1 Emergent Local Knowledge

Emergence is a term that describes how complex systems can arise from relatively simple interactions. Our own mental models of reality are an example of this in that they are the product of simple interwoven observations. Put another way, what you think of when you read the word *dissertation* is the product of millions of neurons responding the symbols on this printed page. Emergence is common in nature and could be the basis for a whole new approach to knowledge management.

In our model, as in the real world, knowledge can originate anywhere in the system. There is a presumption with the current technologies that the creator of this knowledge will place it in a centralized repository such as a database server. Whoever subsequently needs

it, simply gets it from this repository. There is, however, no need for this to be so. It is possible for the knowledge, once it is created, to stay with the creator or with the region to which it pertains or both. This dissertation shows how much information can persist within an area, but it does address what happens when all the agents leave the area. What happens to the information then?

It is possible to build a hybrid between our own work, a peer-to-peer network, and a conventional client-server database management system that would leverage the benefits of emergence of each. Our work provides immediate access to critical spatial information within a framework that is uniquely well suited to support emergence. The P2P approach provides support for collaborative or consensus-based decision-making, while the conventional paradigm ensures long-term data permanence and a measure of central control for certain types of information.

7.3.2 Query Language Modifications

Though much work has been done by others in adapting query languages to the unique opportunities and constraints of mobile environments, our work opens up a large number of new possibilities. For instance, we already stated that when nodes transmit a query, their neighbors immediately start providing it with cached information, even as the remainder of the query finds its way to the remote data server. Under some conditions, nodes may decide that the partial responses they receive from their neighbors are sufficient for their needs. This could be specified in the query itself. Similarly, many other opportunities exist to tailor the information flows to suit the needs of the mobile nodes.

7.3.3 Adaptive Geocasting

Our approach to geocasting, effective as it is, is incapable to adapting to a dynamic environment. If it were, individual nodes would be able to learn from the routing decisions they

(or perhaps their neighbors) have made priorly. This would amount to machine learning, for which the field of game theory provides ample support.

We have already done preliminary work in the application of game theory to improving the efficiency of geocasting, but our crude results are inconclusive. It is very possible to device a feedback mechanism which would allow routing nodes to better estimate when they should relay a particular message so as to maximize its probability of being delivered while minimizing the global cost of so doing.

7.3.4 Security Mechanisms

As we already mentioned in section 6.3.1, there are significant security issues that must be addressed in the realization of our framework. Chief among those would be the adaptation of a lightweight Public Key Infrastructure (PKI) framework to work well within our own. Specifically, we want an adaptive asymmetric encryption environment that minimizes power and computational requirements. Though much work has been done in this area, most of it does not satisfy military-grade security requirements.

It would be interesting and productive to explore ways to use PKI within a lightweight framework such as our own. We are particularly interested in using abbreviated public keys as node identifiers. This would reduce transmission costs while maintaining support for a high level of system integrity. It would also reduce computational (and hence power) requirements. The problem is that we cannot forfeit security for the sake of reducing drain on our batteries.

REFERENCES

[1] Carlisle Adams and Steve Lloyd. *Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations*. Macmillan Technical Publishing, 1999.

[2] J.N. Al-Karaki and A.E. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications Magazine*, 11(6):6–28, 2004.

[3] Virgílio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the www. In *DIS '96: Proceedings of the fourth international conference on on Parallel and distributed information systems*, pages 92–107, Washington, DC, USA, 1996. IEEE Computer Society.

[4] Young bae Ko and Nitin H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, 2004.

[5] Philip A. Bernstein and Nathan Goodman. Concurrency control in distributed database systems. *ACM Comput. Surv.*, 13(2):185–221, 1981.

[6] 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. In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, New York, NY, USA, 1998. ACM.

[7] Stefano Ceri and Giuseppe Pelagatti. *Distributed databases principles and systems*. McGraw-Hill, Inc., New York, NY, USA, 1984.

[8] Chi-Yin Chow, Hong Va Leong, and Alvin T.S. Chan. Grococa: group-based peer-to-peer cooperative caching in mobile environment. *Selected Areas in Communications, IEEE Journal on*, 25(1):179–191, Jan. 2007.

[9] William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979.

[10] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 177–190, New York, NY, USA, 2002. ACM.

[11] Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Cooperative caching: using remote client memory to improve file system performance.

In *OSDI '94: Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 19, Berkeley, CA, USA, 1994. USENIX Association.

[12] Hector Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Trans. Database Syst.*, 8(2):186–213, 1983.

[13] Michael Gerharz and Christian de Waal. Bonnmotion: A mobility scenario generation and analysis tool, April 2009. `http://iv.cs.uni-bonn.de/wg/cs/applications/bonnmotion/`.

[14] Ralf Hartmut Guting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399, 1994.

[15] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, 1984.

[16] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4):287–317, 1983.

[17] Abdelsalam A. Helal, Abdelsalam A. Heddaya, and Bharat K. Bhargava. *Replication Techniques in Distributed Systems*. Springer, Inc., 1996.

[18] Xiaoyan Hong, Mario Gerla, Guangyu Pei, and Ching-Chuan Chiang. A group mobility model for ad hoc wireless networks. In *MSWiM '99: Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 53–60, New York, NY, USA, 1999. ACM.

[19] Fred Howell and Ross Mcnab. simjava: A discrete event simulation library for java. In *In International Conference on Web-Based Modeling and Simulation*, pages 51–56, 1998.

[20] Haibo Hu, Jianliang Xu, Wing Sing Wong, B. Zheng, Dik Lun Lee, and W.-C. Lee. Proactive caching for spatial queries in mobile environments. *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 403–414, April 2005.

[21] Zhiyong Huang, Christian S. Jensen, and Beng Chin Ooi. Collaborative spatial data sharing among mobile lightweight devices. *Advances in Spatial and Temporal Databases*, 4605/2007:366–384, 2007.

[22] Richard Hull. Managing semantic heterogeneity in databases: a theoretical prospective. In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 51–61, New York, NY, USA, 1997. ACM.

[23] Tomasz Imieliński and Julio C. Navas. Gps-based geographic addressing, routing, and resource discovery. *Commun. ACM*, 42(4):86–92, 1999.

[24] Matthias Jarke and Jurgen Koch. Query optimization in database systems. *ACM Comput. Surv.*, 16(2):111–152, 1984.

[25] Deborah G. Johnson. *Computer Ethics*. DIANE Publishing Company, 1998.

[26] Young-Bae Ko and N. H. Vaidya. Geotora: a protocol for geocasting in mobile ad hoc networks. In *Proceedings of the 2000 International Conference on Network Protocols*, page 240, Washington, DC, USA, 2000. IEEE Computer Society.

[27] Young-Bae Ko and Nitin H. Vaidya. Geocasting in mobile ad hoc networks: Location-based multicast algorithms. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 101, Washington, DC, USA, 1999. IEEE Computer Society.

[28] Roger Koenker and Ng Pin. Sparsem: A sparse matrix package for r. *Journal of Statistical Software*, 2003.

[29] Vijay Kumar. *Mobile Database sSystems*. John Wiley and Sons, Inc., Hoboken, NJ, USA, 2006.

[30] Butler Lampson and Howard Sturgis. Crash recovery in a distributed data storage system. *Journal of Internet Technology*, 1979.

[31] Wen-Hwa Liao, Yu-Chee Tseng, Kuo-Lun Lo, and Jang-Ping Sheu. Geogrid: A geocasting protocol for mobile ad hoc networks based on grid. *Journal of Internet Technology*, pages 23–32, 2000.

[32] Christian Maihofer. A survey of geocast routing protocols. *IEEE Communications Surveys and Tutorials*, 6:32–42, 2004.

[33] Walter Maner. *Starter Kit on Teaching Computer Ethics*. Helvetia Press, 1980.

[34] M. L. Markus, A. Majchrzak, and L. Gasser. A design theory for systems that support emergent knowledge processes. *Management Information Systems Quarterly*, 26:179–212, 2002.

[35] Fernando Maymí, Manuel Rodríguez-Martínez, Yi Qian, and Paul C. Manz. Ancile: Pervasively shared situational awareness. *IEEE Internet Computing*, 12(1):48–50, 2008.

[36] Fernando J. Maymí and Paul Manz. Ancile: Dismounted soldier tracking and strike warning. In *Proceedings of the 25th Army Science Conference*, 2006.

[37] Fernando J. Maymí and Manuel Rodríguez-Martínez. Obstacle avoidance for utility-based geocasting. In *Proceedings of the 6th International Conference on Information Technology: New Generations*, 2009.

[38] Julio C. Navas and Tomasz Imielinski. Geocast—geographic addressing and routing. In *MobiCom '97: Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pages 66–76, New York, NY, USA, 1997. ACM.

[39] Fabian Garcia Nocetti, Ivan Stojmenovic, and Jingyuan Zhang. Addressing and routing in hexagonal networks with applications for tracking mobile users and connection rerouting in cellular networks. *IEEE Trans. Parallel Distrib. Syst.*, 13(9):963–971, 2002.

[40] Tamer M. Ozsu and Patrick Valduriez. *Principles of Distributed Database Systems (2nd Edition)*. Prentice Hall, January 1999.

[41] Prasanna Padmanabhan, Le Gruenwald, Anita Vallur, and Mohammed Atiquzzaman. A survey of data replication techniques for mobile ad hoc network databases. *The VLDB Journal*, 17(5):1143–1164, 2008.

[42] Maria Papadopouli and Henning Schulzrinne. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 117–127, New York, NY, USA, 2001. ACM.

[43] E. Pitoura and B. Bhargava. Revising transaction concepts for mobile computing. In *Proceedings of the 1994 workshop on mobile computing systems and applications*, pages 164–168. IEEE, 1994.

[44] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems*. McGraw-Hill, Inc., New York, NY, USA, 2003.

[45] David B. Resnik. What is ethics in research & why is it important?, February 2007. `http://www.niehs.nih.gov/research/resources/bioethics/whatis.cfm`.

[46] Thomas Robertazzi. *Computer Networks and Systems: Queueing Theory and Performance Evaluation*. Springer-Verlag, 2000.

[47] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 71–79, New York, NY, USA, 1995. ACM.

[48] H. Sagan. *Space Filling Curves*. Springer-Verlag, 1994.

[49] Patricia Serrano-Alvarado, Claudia Roncancio, and Michel Adiba. A survey of mobile transactions. *Distributed and Parallel Databases*, 16(2):192–230, 2004.

[50] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.

[51] Ivan Stojmenovic, Anand Prakash Ruhil, and D. K. Lobiyal. Voronoi diagram and convex hull based geocasting and routing in wireless networks: Research articles. *Wirel. Commun. Mob. Comput.*, 6(2):247–258, 2006.

[52] Kishor Shridharbhai Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications (2nd Edition)*. John Wiley and Sons, Inc., 2002.

[53] U.S.C. U.s. code: Title 44, chapter 35, section 3542. definitions, 2008.

[54] Norbert Wiener. *The Human Use of Human Beings: Cybernetics and Society (Da Capo Paperback)*. Da Capo Press, April 1988.

[55] Liangzhong Yin and Guohong Cao. Supporting cooperative caching in ad hoc networks. *Mobile Computing, IEEE Transactions on*, 5(1):77–89, Jan. 2006.

APPENDICES

APPENDIX A

Code Listings

A.1 Mobility Scenario Analysis in R

Listing A.1. LOWESS Regression in R

```
function (n=30, folder="randomway30/")
{
   #define vectors to hold statistics of interest for each sim
   mu = rep(0,n)          #service rates
   lambda = rep(0,n)      #arrival rates
   nir = rep(0,n)         #mean num of nodes in region
   tir = rep(0,n)         #mean node time in region
   nv = rep(0,n)          #mean num of node visits
   for(iteration in 1:n) {
      filename = paste(folder,"/r", iteration,".movements",sep="")
      datadump = readLines(filename)
      N = length(datadump)
      nodesInRegion = 0
      nodesOutside = 0
      nodeTimeInRegion = rep(0,N)
      nodeVisits = rep(0,N)
      for(i in 1:N) {
         #split the single string containing the data line
         data = strsplit(datadump[i], " ")
         #turn the vector of strings from line into numbers
         data = as.numeric(data[[1]])
         it = 1    #index into time coord
         ix = 2    #index into x coord
         iy = 3    #index into y coord
         #if first record read, initialize data structures
         if(i==1) {
            #length of sim in seconds
            simlen=floor(data[length(data)-2])
```

```
    arrivals = rep(0,simlen)
    departures = rep(0,simlen)
    nodes = rep(0,simlen)
}
#figure out if the node is in the region at sim start
if(data[ix]>=1000 && data[iy]>=1000 && data[ix]<2000 &&
        data[iy]<2000) {
    nodesInRegion = nodesInRegion + 1
    inRegion = 1
    nodeTimeInRegion[i] = nodeTimeInRegion[i] + 1
    nodeVisits[i] = nodeVisits[i] + 1
}
else {
    nodesOutside = nodesOutside + 1
    inRegion = 0
}
it = 1   #index into time coord
ix = 2   #index into x coord
iy = 3   #index into y coord
for(s in 1:(length(data)/3-1)) {
   #determine the equation for the first line segment
    deltat = data[it+3]-data[it]
    deltax = data[ix+3]-data[ix]
    deltay = data[iy+3]-data[iy]
    a = deltay/deltax
    b = data[iy]-(a*data[ix])
    ratex = deltax/deltat
    ratey = deltay/deltat   #vertical line case
    x = data[ix]
    y = data[iy]
    #plot points in segment every second
    #and test for region entrance
    for(t in floor(data[it]):(floor(data[it+3])-1)) {
       #test for the case in which the node doesn't move
       if(deltax == 0 && deltay == 0) {
          if(inRegion == 1) {
             nodes[t] = nodes[t] + 1
             nodeTimeInRegion[i] = nodeTimeInRegion[i] + 1
          }
       }
       #otherwise the node is moving
       else {
          #if line is not vertical, use its equation
          if(deltax != 0) {
             y = a*x + b
          }
          #if line is vertical, we manually increase y
          else {
```

```
            y = y + ratey
        }
        #print(y)
        #if the point is in the region...
        if(x>=1000 & y>=1000 & x<2000 & y<2000) {
            #if node wasn't in, register the arrival
            if(inRegion == 0) {
                arrivals[t] = arrivals[t] + 1
                nodes[t] = nodes[t] + 1
                inRegion = 1
                nodeTimeInRegion[i] = nodeTimeInRegion[i] + 1
                nodeVisits[i] = nodeVisits[i] + 1
            }
            #otherwise, node was in, so just record it
            else {
                nodes[t] = nodes[t] + 1
                nodeTimeInRegion[i] = nodeTimeInRegion[i] + 1
            }
        }
        #otherwise, the node is not in the region, so...
        else {
            #if node was in region, record departure
            if(inRegion == 1) {
                departures[t] = departures[t] + 1
                inRegion = 0
            }
        }
        x = x + ratex
      }
    }
    #set up for the next segment
    it = it + 3
    ix = ix + 3
    iy = iy + 3
  }
}
#calculate the service rates and the mean service rate (mu)
MU = rep(0,simlen)
for(i in 1:simlen) {
  if(nodes[i]>0 | departures[i]>0)
    MU[i] = departures[i]/(nodes[i]+departures[i])
}
#unlink("mobilityTestScenario.txt")
mu[iteration]=mean(MU)
lambda[iteration]=mean(arrivals)
nir[iteration]=mean(nodes)
tir[iteration]=mean(nodeTimeInRegion)
nv[iteration]=mean(nodeVisits)
```

```
    }

    #prepare table for export
    results = matrix(0,n,5)
    results[,1] = lambda
    results[,2] = mu
    results[,3] = nir
    results[,4] = tir
    results[,5] = nv
    write.table(paste(folder,"mobility.param.analysis",sep=""))
}
```

A.2 LOWESS Regression in R

Listing A.2. LOWESS Regression in R

```
function() {
        data = read.table("summary_no0.txt")
        col1 = log(data[,1])
        col2 = log(data[,2])

        mod.lo = loess(col3 ~ col1 + col2, span=2)
        print(summary(mod.lo))

        l = seq(min(col1),max(col1), len=25)
        m = seq(min(col2),max(col2), len=25)
        newdata = expand.grid(col1=l, col2=m)
        fit.facts = matrix(predict(mod.lo, newdata),25,25)

        persp(l, m, fit.facts, theta=30, phi=30,
                ticktype="detailed", xlab="lambda",
                ylab="mu", zlab="Facts", shade=0.5)
}
```