# A Cooperative Spatial-Aware Cache for Mobile Environments

Fernando J. Maymí *, Manuel Rodríguez-Martínez ** Wolfgang Andre Rolke ***

*Dept. of Electrical Engineering & Computer Science
U.S. Military Academy
West Point, NY, USA
fernando.maymi@usma.edu

**Dept. of Electrical & Computer Engineering
University of Puerto Rico
Mayaguez, PR, USA
manuel.rodriguez7@upr.edu

***Dept. of Math. Sciences
University of Puerto Rico
Mayaguez, PR, USA
wolfgang.rolke@upr.edu

***Abstract*— In many scenarios, particularly in military and emergency response operations, mobile nodes that are in close proximity to each other exhibit a high degree of data affinity. For example, all soldiers in the same region, regardless of their specialty, will want to know all nearby threats, as well as all friendly assets. Since relaying queries to a distant server is costly in terms of bandwidth and battery power, it would be ideal to use local resources that are only a hop away. In this paper we propose a shared spatial 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. We describe the analytical models that verify our claims and show the results of extensive simulations that validate our models under simulated but realistic conditions.**

## I. Introduction

There exists information that can literally save your life. If you are about to step on a landmine whose existence is known to intelligence officers in their headquarters, then having access to their knowledge will prevent the loss of your life. If you are the pilot of an already-loaded helicopter extracting victims from a flooded area, and you spot an elderly person stranded on a rooftop, then making that information available to anyone who enters that area can help save that victim's life. In both cases, there probably exists a master data store into which the reports, of the landmine and of the victim, are entered. In both cases, personnel in the areas will likely query these data stores for this information. In both cases, delays or interruptions in moving the information from the servers to the mobile clients can be tragic.

Our work 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 called Ancile [1] 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, we extended the scope of the framework to also warn soldiers whenever they approached the vicinity of a suspected improvised explosive device (IED) given that some other soldier already knew about the potential danger.

As we continued extending the architecture to encompass other threats besides 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 [2]. Among these problems is the monitoring and control of emergency response personnel at the site of a man-made or natural disaster. We 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.

A shortcoming of our framework, however, was that we still relied 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.

As we detail in section III, exceptionally good work has

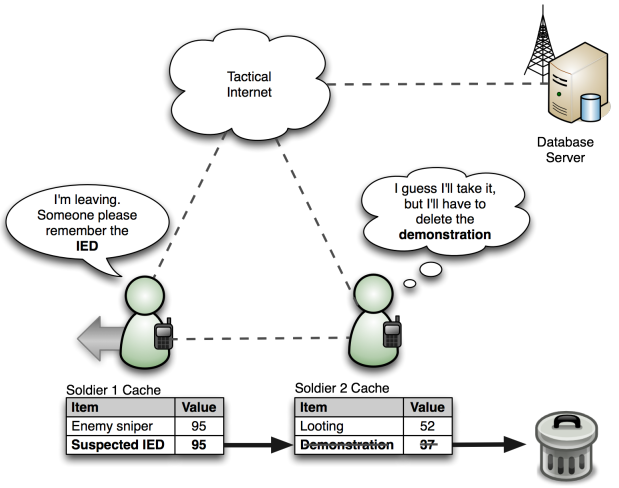


Fig. 1. Cooperative Cache Scenario.

been done in the area of cooperative caching in mobile ad-hoc environments. Significantly, however, the existing body of work does not address a coordinated interaction among the nodes in order to ensure that spatial data concerning a region stays within (or as close as possible to) that region. Nodes in these approaches act altruistically in providing cached data to their neighbors, but do not coordinate with those neighbors their own departures from the area or their deletions of cached data. This means that these cooperative caches are repaired as data is lost (as happens when caching nodes leave), but are not proactively maintained

Our idea, which is sketched at a very high level in figure 1, is to have nodes monitor each other's cache contents as well as their own impending departure from the region for which data is being cached. 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.

### A. 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 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.

### B. 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 paper 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. The specific contributions of this paper are as follows.

1) A framework for distributed, cooperative caching of spatial data in bounded regions of space.
2) An analytical model of the bounds of cooperation among nodes sharing data in a spatial region.
3) An extensive simulation study of the performance of the proposed framework under realistic conditions.

The rest of this paper details our work and is organized as follows. In section II we highlight the essential concepts which the reader will need to understand our work. We follow with a brief survey of relevant research published by others in section III, before we detail our own architecture in section IV. We then delve into a detailed model analysis in section V and a realistic simulation that is documented in section VI. Finally, we draw our conclusions in section VII and provide directions for our future work.

## II. Supporting Concepts

In our paper, we assume the reader is familiar with certain concepts upon which we rely for our framework. In the interest of completeness, we briefly cover these in the following sections. Readers who are familiar with these ideas may safely skip ahead to section III.

### A. Military Grid Reference System

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.

### B. R-trees

Data which pertains to a given geospatial point or region, such as an MGRS grid square, is known as spatial data. Spatial data is not well suited for traditional indexing structures because it is difficult to create a partial order out of a collection of them. One of the most popular approaches to spatial indexing was first proposed by Anton Guttman and is called R-trees [3]. R-trees are similar to the B-trees used in conventional databases in that they are height-balanced trees whose leaf nodes contain pointers to data objects. 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 (MBBs). MBBs are aggregated at each level of the tree, so that all non-leaf nodes are described and indexed by a MBB that encompasses all of the children nodes' geometries.

Figure 2 depicts a hypothetical R-tree index. Note that, for the sake of simplicity, we are combining fixed features (e.g., a train station), with moving objects (e.g., people); it is usually more advisable to not include these two different types of information in the same relation. A critical parameter of an R-tree index is $M$, the maximum number of entries which will fit in one (non-leaf) node. Each internal node other than the root node is required to have $m$ entries, where $\frac{M}{2} \leq m \leq M$. When $m = M$ and a new item is inserted, the node is split into two nodes with roughly the same number of entries each. This change propagates up through the tree an may result in parent nodes being likewise split in two all the way to the root. Conversely, when $m = \frac{M}{2}$ and an item is removed from the tree, that node is deleted and its now-orphaned items will be re-inserted into the tree at the deleted node's parent node.

Indexing in general and R-trees in particular, are applicable to data collections spanning several thousands or even millions of records. While this can realistically be the case for the servers in our framework, we do not envision the need for R-trees in our mobile nodes. The utility of R-trees in our framework stems from ease with which servers that use them can select a given number of records within a spatial region. By defining our regions in terms of R-tree MBBs, we expedite the process of building caches from the server's perspective.

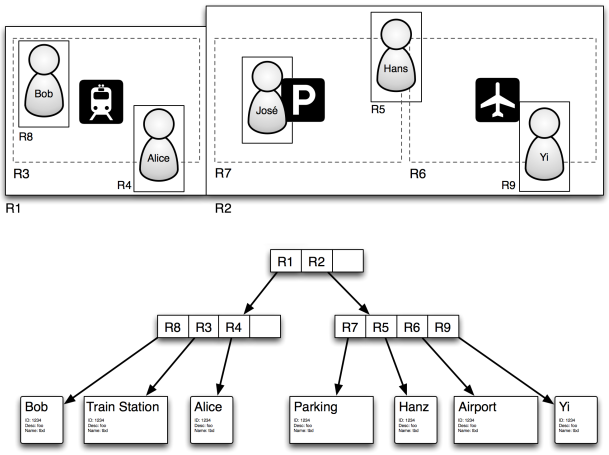


Fig. 2. An R-tree Example.

### C. 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 [4], which shares cached data among directly connected neighbors, and a more flexible approach proposed by Liangshong Yin and Guohong Cao [5], which allows nodes to exploit the caches of others that are not directly connected. We discuss this later approach next, followed by others that are relevant to our proposed framework.

## III. Related Work

Yin and Cao [5] 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. [6], 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. [7], 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.

## IV. 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.

### A. Cache Regions

We divide space into regions using the Military Grid Reference System (MGRS) introduced in section II. 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 VI. 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 in $r$ shared caches. The regions to which these vectors point are then added to the node's collection of caches.

In figure 3, 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 3, 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 3, 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.

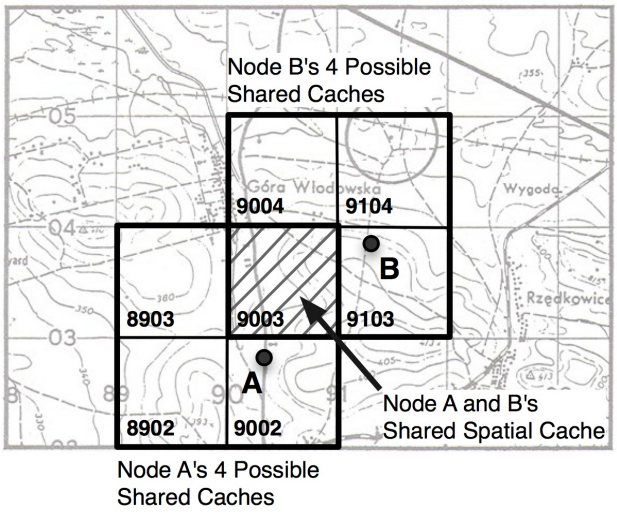


Fig. 3. Shared Spatial Cache Regions

### B. Cache Structure

Each node's cache consists of five lists of pages: one for each of the four spatial caches in which the node may collaborate, and one for all pages that are available for use. Initially, the four 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 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 a node that is caching data about a region leaves that region, it jettisons 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 reset the number of cache hits and number cache copies, but retain the values in the other fields.

### C. 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. 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 it's 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 IV-D, 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 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 2 and 9 to test for a full cache. Lines 1 through 5 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 7 is derrived explicitly in section V-B later in this paper.

**Algorithm 1** Cache Admission Algorithm

**Require:** $cache.length \leq cache.maxlength$
1: **if** $msg.src = SRVR$ and $msg.dst = THIS$ **then**
2: **if** $cache.length = cache.maxlength$ **then**
3: JETTISON()
4: **end if**
5: $cache \leftarrow cache + msg.payload$
6: **else if** $msg.src = SRVR$ or $msg.type = JETTISON$ **then**
7: $p \leftarrow \frac{7.41n+\sqrt{50.9n^2-21.64n}}{2n^2+10.82n}$
8: **if** RANDOM() $\leq p$ **then**
9: **if** $cache.length = cache.maxlength$ **then**
10: JETTISON()
11: **end if**
12: $cache \leftarrow cache + msg.payload$
13: **end if**
14: **end if**

### D. Data Jettison

A key aspect of our architecture is the inclusion of 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. 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.

### E. 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. Therefore, we chose to assign this value using the formula in equation 1.

$$v(i) = \frac{h}{r} \tag{1}$$

This value is computed 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. In general, however, 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

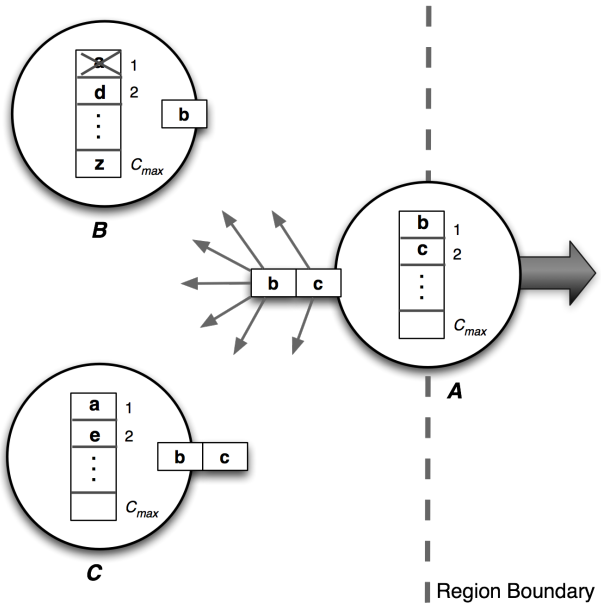


Fig. 4. Data Jettison

replaced with newer ones. This means that the shared spatial cache will encompass a larger area than its data.

The cache replacement, or deletion, process is described in algorithm 2. 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 delete. Note that the test in line 13 is necessary for the case in which the size of the cache is zero.

**Algorithm 2** Replacement Algorithm

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

### F. 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 a previous paper on geocasting by the authors [9]. 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 IV-C 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.

## V. 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.

### A. 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 5. We assumed that nodes arrived at the region at a rate which followed a Poisson distribution with rate $\lambda$. 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 $n\mu$ [10].

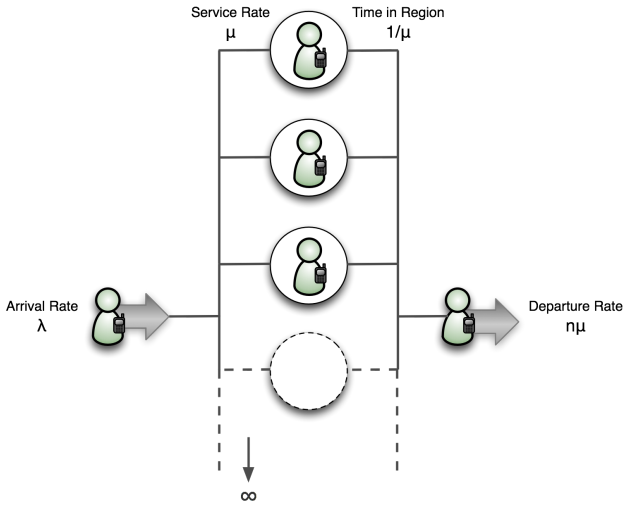


Fig. 5. Spatial Regions as $M/M/\infty$ Queues

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 2.

$$c_{max} < \frac{B}{n\mu S} \tag{2}$$

Now that we knew the maximum contribution of any one node to the shared cache, we wanted to understand the probability that every data item concerning the region was cached 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.

### *B. 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 3.

$$P\left(X < \frac{1-pn}{\sqrt{np(1-p)}}\right) = 0.01 \tag{3}$$

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 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 5.

$$\frac{1-pn}{\sqrt{np(1-p)}} = -2.326 \tag{4}$$

$$p = \frac{7.41n + \sqrt{50.9n^2 - 21.64n}}{2n^2 + 10.82n} \tag{5}$$

As we show in figure 6, 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 7, 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 that same region is $\lambda/\mu$, 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 $\mu^2/n$.

By substituting the expected number of nodes within the region ($\lambda/\mu$) in equation 5 above, we then know that the expected probability for adopting a jettisoned cache item is given by equation 6.

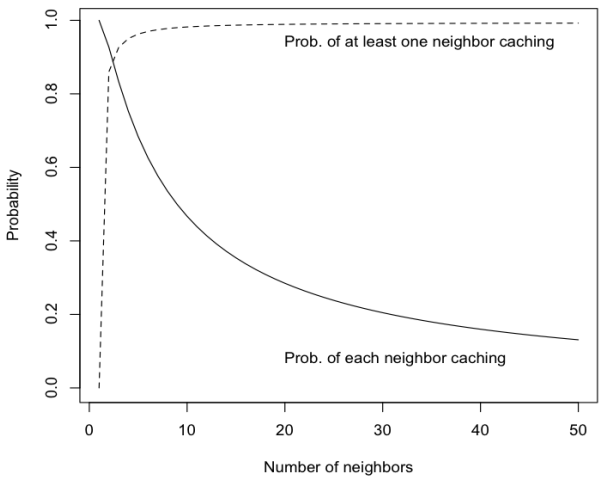


Fig. 6. Probability of item adoption

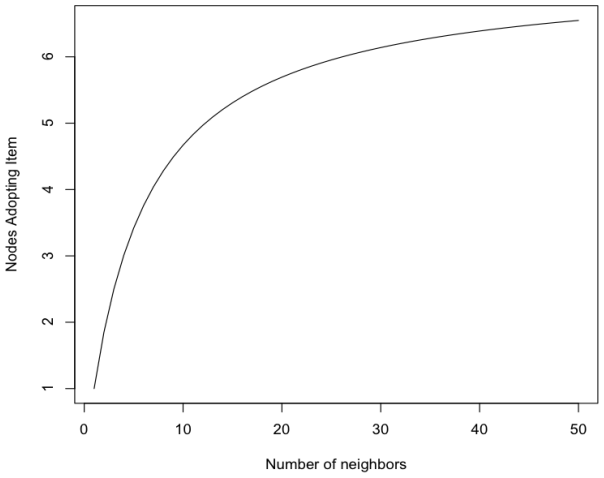


Fig. 7. Number of nodes adopting an item

$$Ep = \frac{7.41\frac{\lambda}{\mu} + \sqrt{50.9\left(\frac{\lambda}{\mu}\right)^2 - 21.64\frac{\lambda}{\mu}}}{2\left(\frac{\lambda}{\mu}\right)^2 + 10.82\frac{\lambda}{\mu}} \quad (6)$$

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 modeled 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 7.

## VI. System Simulations

### A. Estimating Queue Model Parameters

Our entire system model hinges on the parameters of its underlying queue model. The arrival ($\lambda$) and service ($\mu$) 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 $\mu$ 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 $\lambda$ and $\mu$. For this, we made use of BonnMotion, a mobility scenario generation and analysis tool developed at the University of Bonn [11]. With it, we ran six different mobility scenarios using three 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 [12], which is commonly used for modeling unconstrained pedestrian traffic. We also used the Manhattan Grid model [13] to force nodes to travel along streets in a virtual city. Lastly, we used the reference point group mobility model (RPGM) [14], 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 six cases under study and ran the simulations for eight hours each. The results, which are highlighted in table I show that the arrival and service rates for the random waypoint and RPGM models were fairly similar, while those of the Manhattan Grid model were about half their value. This makes sense, seeing how this last model imposes additional constraints on the mobility of the nodes.

Our simulations allowed us to revisit equation 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. 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 II gives us the maximum size of any node's portion of a shared spatial cache under common bandwidths and page sizes.

TABLE I
QUEUE MODEL PARAMETERS

| Model | Parameters | | |
|---|---|---|---|
| | $\lambda$ | $\mu$ | $\bar{n}$ |
| Random Waypoint (30 nodes) | 0.007149306 | 0.001037162 | 6.720866 |
| RPGM (30 nodes) | 0.006862805 | 0.001212537 | 6.508561 |
| Manhattan Grid (30 nodes) | 0.003011179 | 0.0008679334 | 3.409662 |
| Random Waypoint (100 nodes) | 0.02302894 | 0.001051836 | 21.46979 |
| RPGM (100 nodes) | 0.0235813 | 0.001117267 | 22.66135 |
| Manhattan Grid (100 nodes) | 0.01004065 | 0.003837517 | 10.96548 |

TABLE II
ESTIMATED VALUES OF $C_{max}$

| Bandwidth | Page size (in bytes) | | | |
|---|---|---|---|---|
| | 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 |

### B. 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.

### C. 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 V-B 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 IV-E. Additionally, for the shared spatial cache, we ran

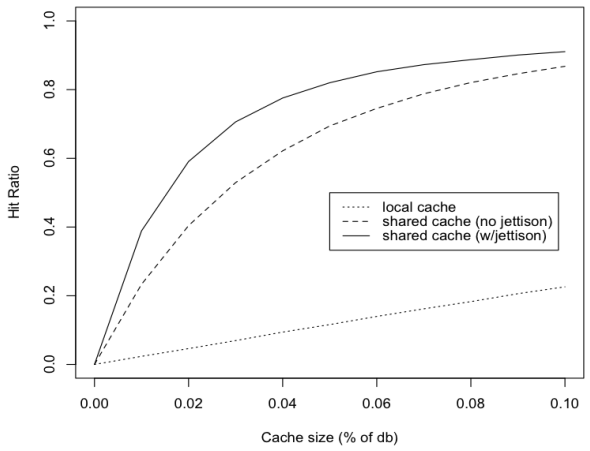


Fig. 8. Cache Hit Ratios

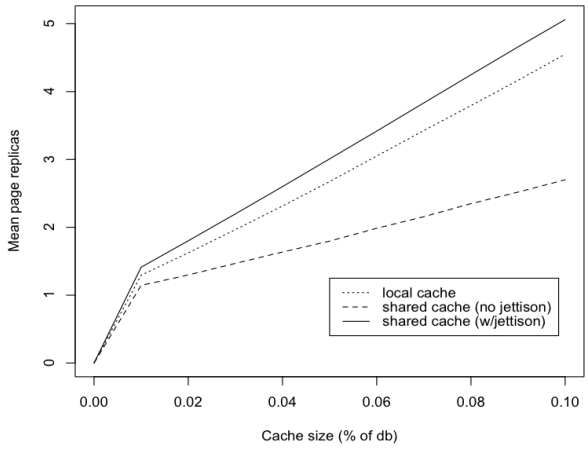


Fig. 9. Number of copies of items in cache

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.

### *D. Experiment Results*

The results of our first set of experiments is depicted in figure 8. 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 30,000 pages pertaining to the central one-square-kilometer region. Each node's cache memory can hold between 300 and 3,000 pages ( 1% to 10% of the total region's pages) in increments of 300 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.

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 9 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.

## VII. Conclusions and Future Work

### *A. Conclusions*

We have presented a novel and innovative method of cooperatively caching spatial data among nodes in a region. Our analytical models showed the conditions under which such a shared spatial cache would be feasible. Based on our experience in both the technical aspects of database systems as well as the tactical aspects of conducting military and emergency response operations, the parameters of our analytical models seem well within the boundaries of realism. Based on those parameters, we conducted detailed simulation studies to determine the performance of our shared spatial caches.

The results of both our analysis and simulations have shown that our approach to caching spatial data in mobile environments 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. These results are, in our opinions, important enough to warrant follow-on development as we describe next.

### B. Future Work

As we noted in our analysis of our results, the overall shared cache performance depends on both the cache size of each constituent node as well as on the total number of nodes within the region. While this is hardly surprising, it does point out an opportunity for improving our model by making the nodes dynamically adjust their cache size in response to both the density of nodes in the region and the density of information about the region that is stored in the server.

Similarly, the fact that mobile nodes have limited power that varies from one node to the other opens an opportunity to study the effects of dynamic cache size selection as a function of available power. A larger cache is more expensive to maintain from a power perspective, because it implies more transmissions in order to support both queries from neighboring nodes, as well as jettisoning as the node leaves the region. Power-aware shared caching is thus a promising line of research for us.

Lastly, we intentionally separated the caching issues from the underlying network transmission in order to be able to derive a model that was more agreeable to detailed analysis. Having accomplished our objective, we would like to combine this effort, with our previous one on probabilistic geocasting in order to study the problem of end-to-end data dissemination in a holistic manner.

## REFERENCES


[1] F. J. Maymí and P. Manz, "Ancile: Dismounted soldier tracking and strike warning," in *Proceedings of the 25th Army Science Conference*, 2006.

[2] F. Maymí, M. Rodríguez-Martínez, Y. Qian, and P. C. Manz, "Ancile: Pervasively shared situational awareness," *IEEE Internet Computing*, vol. 12, no. 1, pp. 48–50, 2008.

[3] A. Guttman, "R-trees: a dynamic index structure for spatial searching," *SIGMOD Rec.*, vol. 14, no. 2, pp. 47–57, 1984.

[4] M. Papadopouli and H. 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*. New York, NY, USA: ACM, 2001, pp. 117–127.

[5] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 1, pp. 77–89, Jan. 2006.

[6] H. Hu, J. Xu, W. S. Wong, B. Zheng, D. L. Lee, and W.-C. Lee, "Proactive caching for spatial queries in mobile environments," *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pp. 403–414, April 2005.

[7] Z. Huang, C. S. Jensen, and B. C. Ooi, "Collaborative spatial data sharing among mobile lightweight devices," *Advances in Spatial and Temporal Databases*, vol. 4605/2007, pp. 366–384, 2007.

[8] C.-Y. Chow, H. V. Leong, and A. T. Chan, "Grococa: group-based peer-to-peer cooperative caching in mobile environment," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 1, pp. 179–191, Jan. 2007.

[9] F. J. Maymí and M. Rodríguez-Martínez, "A probabilistic approach to geocasting queries in mobile spatial databases," in *Proceedings of the 35th International Conference on Very Large Data Bases*, 2009.

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

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

[12] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. 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*. New York, NY, USA: ACM, 1998, pp. 85–97.

[13] J. Markoulidakis, G. Lyberopoulos, and M. Anagnostou, "Traffic model for third generation cellular mobile telecommunication systems," *Wireless Networks*, vol. 4, no. 5, pp. 389–400, September 1998.

[14] X. Hong, M. Gerla, G. Pei, and C.-C. 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*. New York, NY, USA: ACM, 1999, pp. 53–60.