GeoQuorums: Implementing Atomic Memory in Mobile *Ad Hoc* Networks
(Extended Abstract)

Shlomi Dolev[1], Seth Gilbert[2], Nancy A. Lynch[2],
Alex A. Shvartsman[3,2], and Jennifer L. Welch[4]

[1] Department of Computer Science, Ben-Gurion University, dolev@cs.bgu.ac.il
[2] MIT CSAIL, {sethg,lynch}@theory.lcs.mit.edu
[3] Department of Computer Science and Engineering, University of Connecticut, alex@theory.lcs.mit.edu
[4] Department of Computer Science, Texas A&M University, welch@cs.tamu.edu

Abstract. We present a new approach, the GeoQuorums approach, for implementing atomic read/write shared memory in *ad hoc* networks. Our approach is based on abstract nodes associated with certain geographic locations. We assume the existence of *focal points*, geographic areas that are normally "populated" by mobile hosts. For example, a focal point may be a road junction, a scenic observation point, or a water resource in the desert. Mobile hosts that happen to populate a focal point participate in implementing shared atomic `put`/`get` objects, using a replicated state machine approach. These objects are then used to implement atomic read/write operations. The GeoQuorums algorithm defines certain intersecting sets of focal points, known as quorums. The quorum systems are used to maintain the consistency of the shared memory. We present a mechanism for changing quorum systems on the fly, thus improving efficiency. Overall, the new GeoQuorums algorithm efficiently implements read and write operations in a highly dynamic, mobile network.

1 Introduction

In this paper, we introduce a new approach to designing algorithms for mobile *ad hoc* networks. An *ad hoc* network uses no pre-existing infrastructure, unlike cellular networks that depend on fixed, wired base stations. Instead, the network

⋆ This work is supported in part by NSF grant CCR-0098305 and NSF ITR Grant 0121277. Part of the work of the first author has been done during visits to MIT and Texas A&M. The first author is partially supported by an IBM faculty award, the Israeli ministry of defense, NSF, and the Israeli Ministry of Trade and Industry. The second and third authors are partially supported by AFOSR Contract #F49620-00-1-0097, DARPA Contract #F33615-01-C-1896, NSF Grant 64961-CS, NTT Grant MIT9904-12. The fourth author is partially supported by the NSF Grant 9988304, 0311368 and by the NSF Career Award 9984774. The fifth author is partially supported by NSF Grant 0098305 and Texas Advanced Research Program 000512-0091-2001.

F.E. Fich (Ed.): DISC 2003, LNCS 2848, pp. 306–320, 2003.

is formed by the mobile nodes themselves, which cooperate to route communication from sources to destinations.

Ad hoc communication networks are, by nature, highly dynamic. Mobile hosts are often small devices with limited energy that spontaneously join and leave the network. As a mobile host moves, the set of neighbors with which it can directly communicate may change completely. The nature of *ad hoc* networks makes it challenging to solve the standard problems encountered in mobile computing, such as location management (e.g., [1]), using classical tools. The difficulties arise from the lack of a fixed infrastructure to serve as the backbone of the network. In this paper, we begin to develop a new approach that allows existing distributed algorithms to be adapted for highly dynamic *ad hoc* environments.

Providing atomic [2] (or linearizable [3]) read/write shared memory in *ad hoc* networks is a fundamental problem in distributed computing. Atomic memory is a basic service that facilitates the implementation of many higher-level algorithms. For example, one might construct a location service by requiring each mobile host to periodically write its current location to the memory. Alternatively, a shared memory could be used to collect real-time statistics, for example, recording the number of people in a building. We present here a new algorithm for atomic multi-writer/multi-reader memory in mobile, *ad hoc* networks.

The GeoQuorums Approach. We divide the problem of implementing atomic read/write memory into two parts. First, we define a static, abstract system model that associates abstract nodes with certain fixed geographic locales. The mobile hosts implement this model using a replicated state machine approach. In this way, the dynamic nature of the *ad hoc* network is masked by a static model. Second, we present an algorithm to implement atomic memory using the static network model.

The geographic model specifies a set of physical regions, known as *focal points*. The mobile hosts within a focal point cooperate to simulate a single virtual process. Each focal point is required to support a local broadcast service, which provides reliable, totally ordered broadcast. This service allows each node in the focal point to communicate reliably with every other node in the focal point. The local broadcast service is used to implement a type of replicated state machine, one that tolerates joins and leaves of mobile hosts. If every mobile host leaves the focal point, the abstract node fails.

The atomic memory algorithm is implemented on top of the geographic abstraction. Nodes implementing the atomic memory algorithm use a GeoCast service (as in [4,5]) to communicate with the virtual processes, that is, with the focal point nodes. In order to achieve fault tolerance and availability, the algorithm replicates the read/write shared memory at a number of focal points. In order to maintain consistency, accessing the shared memory requires updating certain sets of focal points, known as quorums [6,7,8,9,10]. (Note that the members of our quorums are focal points, not mobile hosts.) The algorithm uses two sets of quorums: (i) get-quorums, and (ii) put-quorums, with the property that

every get-quorum intersects every put-quorum.[1] The use of quorums allows the algorithm to tolerate the failure of a limited number of focal points.

Our atomic memory algorithm uses a Global Position System (GPS) time service, allowing it to process writes using a single phase; prior single-phase write algorithms made other strong assumptions, for example, relying either on synchrony [8] or single writers [9]. Our algorithm also allows for some reads to be processed using a single phase: the atomic memory algorithm flags the completion of a previous read or write to avoid using additional phases, and propagates this information to various focal points. As far as we know, this is an improvement on previous quorum-based algorithms.

For performance reasons, at different times it may be desirable to use different sets of get-quorums and put-quorums. For example, during periods of time when there are many more read operations than write operations, it may be preferable to use smaller, more geographically distributed, get-quorums that are fast to communicate with, and larger put-quorums that are slower to access. If the operational statistics change, it may be useful to reverse the situation. The algorithm presented here includes a limited reconfiguration capability: it can switch between a finite number of predetermined configurations. As a result of the static underlying model, in which focal points neither join nor leave, this is not a severe limitation. The resulting reconfiguration algorithm, however, is quite efficient compared to prior reconfigurable atomic memory algorithms [11, 12]. Reconfiguration does not significantly delay read or write operations, and, as no consensus service is required, reconfiguration terminates rapidly.

This paper contains three primary contributions. First, we introduce the geographic abstraction model, which allows simple, static algorithms to be adapted for highly dynamic environments. Second, we provide an implementation of the abstract model using mobile hosts. Third, we implement a reconfigurable, atomic read/write shared memory, using the static model.

Other Approaches. Quorum systems are widely used to implement atomic memory in static distributed systems [6,7,8,9,13,14]. More recent research has pursued application of similar techniques to highly dynamic environments, like *ad hoc* networks. Many algorithms depend on reconfiguring the quorum systems in order to tolerate frequent joins and leaves and changes in network topology. Some of these [15,16,14,10] require the new configurations to be related to the old configurations, limiting their utility in *ad hoc* networks. Englert and Shvartsman [17] showed that using any two quorum systems concurrently preserves atomicity during more general reconfiguration. Recently, Lynch and Shvartsman introduced RAMBO [11] (extended in [12]), an algorithm designed to support distributed shared memory in a highly dynamic environment. The RAMBO algorithms allow arbitrary reconfiguration, supporting a changing set of (potentially mobile) participants. The GeoQuorums approach handles the dynamic aspects of the network by creating a geographic abstraction, thus simplifying the atomic

[1] These are often referred to as read-quorums and write-quorums; the put/get terminology more accurately describes the operations performed on the focal points in the quorums, since read operations may use both types of quorums.

memory algorithm. While prior algorithms use reconfiguration to provide fault tolerance in a highly dynamic setting, the GeoQuorums approach depends on reconfiguration primarily for performance optimization. This allows a simpler, and therefore more efficient, reconfiguration mechanism.

Haas and Liang [18] also address the problem of implementing quorum systems in a mobile network. Instead of considering reconfiguration, they focus on the problem of constructing and maintaining quorum systems for storing location information. Special participants are designed to perform administrative functions. Thus, the backbone is formed by unreliable, *ad hoc* nodes that serve as members of quorum groups. Stojmenovic and Pena [19] choose nodes to update using a geographically aware approach. They propose a heuristic that sends location updates to a north-south column of nodes, while a location search proceeds along an east-west row of nodes. Note that the north-south nodes may move during the update, so it is possible that the location search may fail. Karumanchi *et al.* [20] focus on the problem of efficiently utilizing quorum systems in a highly dynamic environment. The nodes are partitioned into fixed quorums, and every operation updates a randomly selected group, thus balancing the load.

Document Structure. The rest of the paper is organized as follows. The system model appears in Section 2. The algorithms for emulating a focal point and implementing GeoQuorums appear in Section 3. The atomicity proof for the implementations appear in Section 4. Section 5 contains a discussion of the performance of the algorithm. Finally, in Section 6, we conclude and present some areas for future research. The complete code for the algorithms and selected proofs are given in the full technical report [21].

2 System Model

In this section, we describe the underlying theoretical model, and discuss the practical justifications.

Theoretical Model. Our world model consists of a bounded region of a two-dimensional plane, populated by mobile hosts. The mobile hosts may join and leave the system, and may fail at any time. (We treat leaves as failures.) The mobile hosts can move on any continuous path in the plane, with bounded speed. The computation at each mobile host is modeled by an asynchronous automaton, augmented with a *geosensor*. The geosensor is a device with access to a real-time clock and the current, exact location of the mobile host in the plane. It provides the mobile host with continuous access to this information.

While we make no assumption about the motion of the mobile hosts, we do assume that there are certain regions that are usually "populated" by mobile hosts. We assume that there is a collection of some n uniquely identified, non-intersecting regions in the plane, called *focal points*, such that (i) at most f focal points fail (for some $f < n$), in the sense that there is a period of time during which no mobile host is in the focal point region, and (ii) the mobile hosts in each focal point are able to implement a reliable, atomic broadcast service. Condition (i) is used to ensure that sufficiently many focal points remain available. Once a focal point becomes unavailable due to "depopulation", we do not allow it

to recover if it is repopulated. (The algorithm we present in this paper can be modified to allow a "failed" focal point to recover, however, we do not discuss this modification here.) Condition (ii) ensures that all mobile hosts within a focal point can communicate reliably with each other, and that messages are totally ordered. We assume that each mobile host has a list of all the focal point identifiers.

Each mobile host also has a finite list of *configurations*. A configuration, c, consists of a unique identifier and two sets of quorums: *get-quorums*(c) and *put-quorums*(c). Each quorum consists of a set of focal points identifiers, and they have the following intersection properties: if $G \in$ *get-quorums*(c) and $P \in$ *put-quorums*(c), then $G \cap P \neq \emptyset$. Additionally, for a given c, we assume that for any set of f focal points, F, there exist $G \in$ *get-quorums*(c) and $P \in$ *put-quorums*(c) such that $F \cap G = \emptyset$ and $F \cap P = \emptyset$. This allows an algorithm based on the quorums to tolerate f focal points failing. Fur the purposes of this presentation, we assume there are only two configurations, c_1 and c_2.

Mobile hosts depend on two broadcast services: (i) LBCast, a local, atomic broadcast service, and (ii) GeoCast, a global delivery service. The LBCast service allows nodes within a focal point to communicate reliably. Each focal point is assumed to support a separate LBCast service: if we refer to focal point h, its broadcast service is referred to as $lbcast_h$. The LBCast service takes one parameter, a message, and delivers it to every node in the focal point region. If mobile host i is in focal point h, and broadcasts a message m using $lbcast_h$ at time t, and if j is also in focal point h at time t, and remains in h, then j receives message m. Additionally, the service guarantees that all mobile hosts receive all messages in the same order. That is, if host i_1 receives message m_1 before message m_2, then if host i_2 receives messages m_1 and m_2 it will receive message m_1 before message m_2.

The GeoCast service delivers a message to a specified destination in the plane, and optionally delivers it to a specified node at that location. The GeoCast service takes three parameters: (i) message, (ii) destination location, (iii) ID of a destination node (*optional*). If no destination ID is specified, then the destination location must be inside some focal point, h. In this case, if message m is GeoCast at time t, then there exists some time $t' > t$ such that if mobile host i is in focal point h at time t', and remains in h, then i receives message m. If a destination-ID is specified, and if the destination node remains near the destination location until the message is delivered, and the destination node does not fail until the message is delivered, then the service will eventually deliver the message to the node with the correct destination-ID.

Practical Aspects. This theoretical model represents a wide class of real mobile systems. First, there are a number of ways to provide location and time services, as represented by the geosensor. GPS is perhaps the most common means, but others, like Cricket [22], are being developed to remedy the weaknesses in GPS, such as the inability to operate indoors. Our algorithms can tolerate small errors in the time or location, though we do not discuss this.

Second, the broadcast services specified here are reasonable. If a focal point is small enough, it should be easy to ensure that a single broadcast, with appropriate error correction, reaches every mobile node at the focal point. If the broadcast service uses a time-division/multiple-access (TDMA) protocol, which allocates each node a time slot in which to broadcast, then it is easy to determine a total ordering of the messages. A node joining the focal point might use a separate *reservation channel* to compete for a time slot on the main TDMA *communication channel*. This would eliminate collisions on the main channel, while slightly prolonging the process of joining a focal point.

The GeoCast service is also a common primitive in mobile networks: a number of algorithms have been developed to solve this problem, originally for the internet protocol [4] and later for *ad hoc* networks (e.g., [23,5]).

We propose one set of configurations that may be particularly useful in practical implementations. We take advantage of the fact that accessing nearby focal points is usually faster than accessing distant focal points. The focal points can be grouped into clusters, using some geographic technique [24]. Figure 1 illustrates the relationship among mobile hosts, focal points, and clusters. For configuration c_1, the *get-quorums* are defined to be the clusters. The *put-quorums* consist of every set containing one focal point from each cluster. Configuration c_2 is defined in the opposite manner. Assume, for example, that read operations are more common than write operations (and most read operations only require one phase). Then, if the clusters are relatively small and are well distributed (so that every mobile host is near to every focal point in some cluster), then configuration c_1 is quite efficient. On the other hand, if write operations are more common than read operations, configuration c_2 is quite efficient. Our algorithm allows the system to switch safely between two such configurations.

Another difficulty in implementation might be agreeing on the focal points and ensuring that every mobile host has an accurate list of all the focal points and configurations. Some strategies have been proposed to choose focal points: for example, the mobile hosts might send a token on a random walk, to collect

information on geographic density [25]. The simplest way to ensure that a mobile host has access to a list of focal points and configurations is to depend on a centralized server, through transmissions from a satellite or a cell-phone tower. Alternatively, the GeoCast service itself might facilitate finding other mobile hosts, at which point the definitive list can be discovered.

3 Focal Point Emulator and Operation Manager

The GeoQuorums algorithm consists of two components: the Focal Point Emulator (FPE) and the Operation Manager (OM). Figure 2 describes the relationships among the different components of the algorithm.

For example, a client at some node i may request a read (the "read" arrow from the Client to the OM). The OM notes the mobile host's current location, using the Geosensor (right "geo-update" arrow). The OM then sends GeoCast messages to focal points ("geoc-send" arrow), including its current location. The GeoCast message is received by the FPE at some other node, j, ("geoc-rcv"

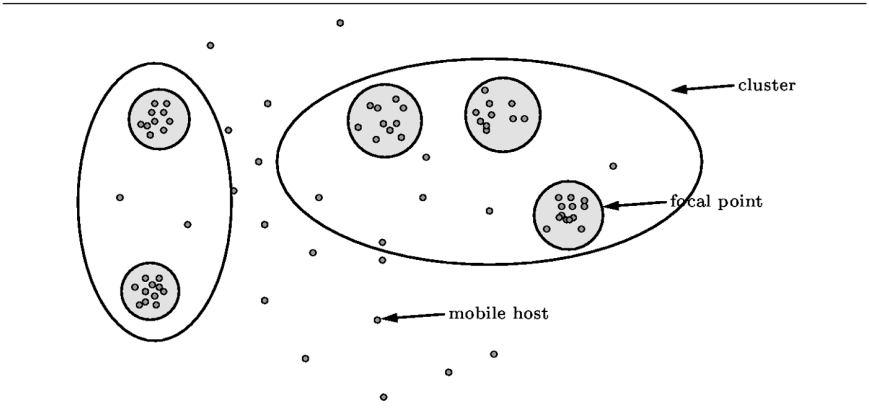

Fig. 1. Clusters

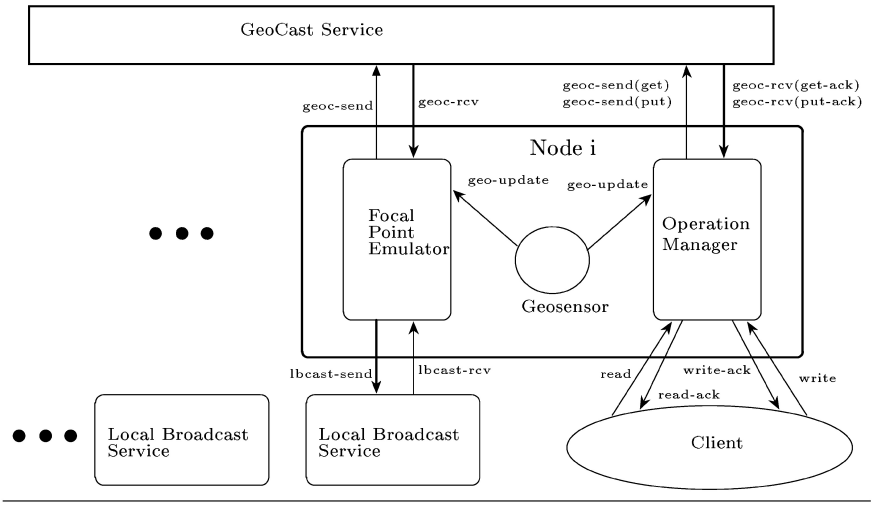

Fig. 2. System Architecture

arrow). The FPE at j first sends a local broadcast of the request ("lbcast-send" arrow), and then sends a response to i ("geoc-send" arrow), using the position of the client received in the GeoCast message. The OM at i uses the responses received ("geoc-rcv" arrow) from the FPEs to compute the response to the read operation, which it sends to the Client ("read-ack" arrow).

A FPE determines that the mobile host is in a focal point region using information from the Geosensor (left "geo-update" arrow). Then the FPE uses the LBCast service to perform the join protocol ("lbcast-send", "lbcast-rcv"),

Signature:

Input:	Output:	Internal:
lbcast-rcv$(message, payload, op\text{-}src)_{h,i}$	lbcast-send$(message, payload, op\text{-}src)_{h,i}$	join$()_i$
geoc-rcv$(message, payload, src, dest)_i$	geoc-send$(message, payload, src, dest)_i$	leave$()_i$
geo-update$(current\text{-}loc, new\text{-}time)_i$		

State Components

$status \in \{\text{idle}, \text{joining}, \text{active}\}$, initially $\begin{cases} \text{active} & \text{if } i \text{ is initially in some focal point} \\ \text{idle} & \text{otherwise} \end{cases}$

join-oid, join id, initially 0	*queues*, a record with fields:
focalpoint-id, focal point id, initially $\perp$	*geocast*, a queue of $\langle op, payload, src, dest\rangle$, initially $\emptyset$
complete-ops, a set of operation ids, initially $\emptyset$	*lbcast*, a queue of $\langle op, payload, op\text{-}src\rangle$, initially $\emptyset$
data, a record with fields:	*global*, a record with fields:
value, a value, initially v_0	*fp-map*, a set of focal points
tag, a tag id, initially $\perp$	*clock*, a clock, initially 0
confirmed, a set of tag ids, initially $\emptyset$	*location*, a location, initially i's initial loc.
conf-id, a configuration id, initially $\langle 0, 0, 0\rangle$	
recon-ip, a boolean flag, initially false	

Fig. 3. Focal Point Emulator FPE_i Signature and State

after which point it can respond to GeoCast messages. We now describe the algorithm in more detail.

3.1 Focal Point Emulator

The Focal Point Emulator (FPE) is the automaton that allows the members of a focal point to simulate a single replica. The FPE implements a replicated state machine, using the totally ordered local broadcast to ensure consistency. Figure 3 contains the signature and state of the FPE. The remaining code for the FPE is available in the technical report [21].

The FPE maintains a *data* record that represents the state replicated at every mobile host in the focal point. The FPE receives put and get requests from the GeoCast service, updating and retrieving *data.value*. Each put is accompanied by a unique tag from a totally ordered set, which is stored in *data.tag*. Occasionally the FPE is notified that a tag is confirmed; *data.confirmed* tracks the set of confirmed tags. (This means that at least one operation involving this tag has fully completed.) Requests to the FPE contain the id of a configuration; *data.conf-id* stores the largest known configuration id. *data.recon-ip* is a flag that indicates whether a reconfiguration is in progress.

The FPE receives various messages from the GeoCast service, sent by mobile hosts. Each incoming message is immediately rebroadcast, using the LBCast service. The FPE takes no other action in response to GeoCast messages.

The FPE also receives messages from the LBCast service. Each FPE automaton can be idle, joining, or active. If a node is not idle (even if it is in the process of joining), then it will process incoming messages and update its local state, in order to maintain consistency. If the node is active (and joining is completed), then the FPE enqueues a response, if required. Finally, if any node

notices that a new configuration is being used, it sets a flag to remember that a reconfiguration is in progress.

The LBCast service delivers four types of messages: (i) If FPE_i (the FPE at node i) receives a get message and no other node has responded, then FPE_i sends a response via the GeoCast service, containing its current *data.tag*, *data.value*, and *data.confirmed*. (ii) If FPE_i receives a put message, then it updates its *data.tag*, *data.value* and *data.confirmed* using the data in the message. If no other node has responded, then FPE_i sends a response using the GeoCast service, indicating that the update is complete. (iii) If FPE_i receives a confirm message, then it updates its local copy of the confirmed flag. (iv) If node i receives a *recon-done* message, then it sets its local *recon-ip* flag to *false* to indicate that the reconfiguration is completed.

The final piece of the FPE is the join protocol, which enables a mobile host to join a focal point. Recall that the Geosensor service periodically notifies the mobile host of its new location. When the host has entered a focal point, it begins the join protocol by sending a join-request message using the LBCast service; this message contains a unique identifier for the join request consisting of the requester's node identifier and the current time. When node i receives the join-request message, if no other node has responded, then node i sends a response using the LBCast; this response includes *data.tag*, *data.value*, and *data.confirmed*. As soon as the initiator of the join protocol receives any response, it updates its current *data.value*, *data.tag*, and *data.confirmed* with the information in the response message, and then becomes active.

3.2 Operation Manager

The Operation Manager (OM) maintains the state described in Figure 4. The OM uses the GeoCast service to communicate with FPEs (see Figure 5), sending get, put, and confirm messages, and receiving appropriate responses. The OM

Signature:

Input:	Output:	Internal:
read()$_i$	read-ack($value$)$_i$	read-2()$_i$
write($value$)$_i$,	write-ack()$_i$	recon-2()
recon($config\text{-}name$)$_i$	recon-ack()$_i$	confirm()$_i$
geoc-rcv(op, $payload$, src, $dest$)$_i$	geoc-send(op, $payload$, src, $dest$)$_i$	

State Components

confirmed, a set of tag ids, initially $\emptyset$	*op*, a record with fields:
conf-id, a configuration id, initially $\langle 0, 0, 0 \rangle$	*type* $\in$ {read, write, recon}
recon-ip, a boolean flag, initially false	*phase* $\in$ {idle, get, put}, initially idle
G_1, P_2, G_2, P_2, the sets of get-quorums and put-quorums for configurations 1 and 2	*tag*, a tag id, initially $\perp$
global, a record with fields:	*value*, a value, initially $\perp$
location, a location, initially $\perp$	*recon-ip*, a boolean flag, initially false
clock, a time, initially 0	*oid*, an operation id, initially 0
fp-map, a set of focal point definitions	*acc*, a set of process ids, initially $\emptyset$
	loc, a location, initially $\perp$

Fig. 4. Operation Manager OM_i Signature and State

Input geoc-rcv(op-ack, oid, tag, val, $conf$, cid, $rec\text{-}ip$, src, $dest$)$_i$
Effect:
  if $op.oid = oid$ then
    if op-ack = get-ack and $tag > op.tag$ then
      $op.tag \leftarrow tag$
      $op.val \leftarrow val$
    $acc \leftarrow acc \cup \{lookup(src.loc, global.fp\text{-}map)\}$
  if $cid > conf\text{-}id$ then
    $conf\text{-}id \leftarrow cid$
    $op.recon\text{-}ip \leftarrow$ true
    $recon\text{-}ip \leftarrow$ true
    if $op.type =$ recon then
      $op.phase =$ idle
  else if $cid = conf\text{-}id$ then
    if $rec\text{-}ip =$ false then $recon\text{-}ip \leftarrow$ false
  if $conf =$ true then
    $confirmed \leftarrow confirmed \cup \{tag\}$

Output geoc-send($message$, $payload$, src, $dest$)$_i$
Precondition:
  if ($op.phase \neq$ idle) then
    $message = op.phase \vee$
      $message \in$ {confirm, recon-done}
  else
    $message \in$ {confirm, recon-done}
  $payload$ contains operation specific information (i.e., $data.tag$, $data.value$, etc.)
  $src = \langle i, global.location \rangle$
  $fp\text{-}name \in FP$
  $dest = \langle$focal-point, $fp\text{-}name\rangle$
Effect:
  None

Fig. 5. Operation Manager OM_i GeoCast Send/Receive

uses the FPEs as replicas, guaranteeing both atomicity and fault tolerance. For each phase of each operation, the OM receives messages from a quorum of FPEs.

Read/Write Operations. The code for read/write operations is presented in Figure 6. When OM_i receives a write request, it examines its clock to choose a tag for the operation. OM_i uses the GeoCast service to send the new tag and new value to a number of focal points. Let c be the value of $conf\text{-}id_i$ when the operation begins. If all responses indicate that c is the most recent configuration (i.e., no reconfiguration is in progress), then the operation terminates when OM_i receives at least one response from each focal point in some $P \in put\text{-}quorums(c)$. If any response indicates that a reconfiguration is in progress, then OM_i waits until it also receives responses from each focal point in some $P' \in put\text{-}quorums(c')$, where c' is the other configuration. (We have assumed there are only two configurations – if there are more than two configurations, OM_i would need to hear from all of them.) After the operation is complete, OM_i can optionally notify focal points that the specified tag has been *confirmed*, indicating that the operation is complete.

When OM_i receives a read request, it sends out messages to a number of focal points. Let c be the value of $conf\text{-}id_i$ when the operation begins. As for write operations, if all responses indicate that c is the most recent configuration, then the first phase terminates when OM_i receives a response from each focal point in some $G \in get\text{-}quorums(c)$. Otherwise, the phase completes when OM_i also receives a response from each focal point in some $G' \in get\text{-}quorums(c')$, where c' is the other configuration. At this point, OM_i chooses the value associated with the largest tag from any of the responses. If the chosen tag has been *confirmed*, then the operation is complete. Otherwise, OM_i begins a second phase that is identical to the protocol of the write operation.

Notice that the knowledge of the *confirmed* tags is used to short-circuit the second phase of certain read operations. The second phase is only required in the case where a prior operation with the same tag has not yet completed. By

Input $\mathsf{read}()_i$
Effect:
$op \leftarrow \langle \mathsf{read}, \mathsf{get}, \perp, \perp, recon\text{-}ip, \langle global.clock, i\rangle, \emptyset, global.location\rangle$

Output $\mathsf{read\text{-}ack}(v)_i$
Precondition:
$conf\text{-}id = \langle c, p, n\rangle$
if $op.recon\text{-}ip$ then
$\exists p_0 \in P_0, p_1 \in P_1$ such that $acc \supseteq p_0 \cup p_1$
else
$\exists p_n \in P_n$ such that $acc \supseteq p_n$
$op.phase = \mathsf{put}$
$op.type = \mathsf{read}$
$v = op.value$
Effect:
$op.phase \leftarrow \mathsf{idle}$
$confirmed \leftarrow confirmed \cup \{op.tag\}$

Output $\mathsf{read\text{-}ack}(v)_i$
Precondition:
$conf\text{-}id = \langle c, p, n\rangle$
if $op.recon\text{-}ip$ then
$\exists g_0 \in G_0, g_1 \in G_1$ such that $acc \supseteq g_0 \cup g_1$
else $\exists g_n \in G_n$ such that $acc \supseteq g_n$
$op.phase = \mathsf{get}$
$op.type = \mathsf{read}$
$op.tag \in confirmed$
$v = op.value$
Effect:
$op.phase \leftarrow \mathsf{idle}$

Input $\mathsf{write}(v)_i$
Effect:
$op \leftarrow \langle \mathsf{write}, \mathsf{put}, \langle global.clock, i\rangle, v, recon\text{-}ip, \langle global.clock, i\rangle, \emptyset, global.location\rangle$

Internal $\mathsf{read\text{-}2}()_i$
Precondition:
$conf\text{-}id = \langle c, p, n\rangle$
if $op.recon\text{-}ip$ then
$\exists g_0 \in G_0, g_1 \in G_1$ s.t. $op.acc \supseteq g_0 \cup g_1$
else
$\exists g_n \in G_n$ such that $acc \supseteq g_n$
$op.phase = \mathsf{get}$
$op.type = \mathsf{read}$
$op.tag \notin confirmed$
Effect:
$op.phase \leftarrow \mathsf{put}$
$op.recon\text{-}ip \leftarrow \mathsf{false}$
$op.oid \leftarrow \langle global.clock, i\rangle$
$op.acc \leftarrow \emptyset$
$op.loc \leftarrow my\text{-}location$

Output $\mathsf{write\text{-}ack}()_i$
Precondition:
$conf\text{-}id = \langle c, p, n\rangle$
if $op.recon\text{-}ip$ then
$\exists p_0 \in P_0, p_1 \in P_1$ such that $acc \supseteq p_0 \cup p_1$
else
$\exists p_n \in P_n$ such that $acc \supseteq p_n$
$op.phase = \mathsf{put}$
$op.type = \mathsf{write}$
Effect:
$op.phase \leftarrow \mathsf{idle}$
$confirmed \leftarrow confirmed \cup \{op.tag\}$

Fig. 6. Operation Manager OM_i Read/Write Transitions

notifying focal points when the tag has been confirmed, the algorithm allows later operations to discover that a second phase is unnecessary.

Reconfiguration. The code for the reconfiguration algorithm is presented in Figure 7. The reconfiguration algorithm is a variant of the reconfiguration mechanism presented in the RAMBO II algorithm [12]: the presented algorithm is a special case of the general algorithm, in which there are only a small, finite number of legal configurations. This simplification obviates the need for a consensus service, and therefore significantly improves efficiency. A reconfiguration operation is similar to a read or write operation, in that it requires contacting appropriate quorums of focal points from the two different configurations, c_1 and c_2. First, OM_i determines a new, unique, configuration identifier, by examining the local clock, its node id, and the name of the desired configuration. Then OM_i sets a flag, indicating that a reconfiguration is in progress. At this point, the first phase of the reconfiguration begins: OM_i sends messages to a number of focal points. The first phase terminates when OM_i receives a response from every node in four different quorums: (i) a get-quorum of c_1, (ii) a get-quorum of c_2, (iii) a put-quorum of c_1, and a (iv) put-quorum of c_2. Then the second phase begins, again sending out messages to focal points. It terminates when OM_i re-

Input recon$(conf\text{-}name)_i$
Effect:
  $conf\text{-}id = \langle global.clock, i, conf\text{-}name \rangle$
  $recon\text{-}ip = \text{true}$
  if $op.type = \text{recon}$ then
    $op.phase = \text{idle}$

Internal recon-upgrade-2$(cid)_i$
Precondition:
  $\exists g_0 \in G_0, g_1 \in G_1, p_0 \in P_0, p_1 \in P_1$
    such that: $acc \supseteq g_0 \cup g_1 \cup p_0 \cup p_1$
  $op.type = \text{recon}$
  $op.phase = \text{get}$
  $cid = conf\text{-}id$

Effect:
  $op.phase \leftarrow \text{put}$
  $op.oid \leftarrow \langle global.clock, i \rangle$
  $op.acc \leftarrow \emptyset$
  $op.loc \leftarrow global.location$

Internal recon-upgrade$(cid)_i$
Precondition:
  $recon\text{-}ip = \text{true}$
  $op.phase = \text{idle}$
  $cid = conf\text{-}id$
Effect:
  $op \leftarrow \langle \text{recon}, \text{get}, \perp, \perp, \text{true}, \langle global.clock, i \rangle, \emptyset, global.location \rangle$

Output recon-ack$(cid)_i$
Precondition:
  $conf\text{-}id = \langle c, p, n \rangle$
  $\exists p_n \in P_n$ such that $acc \supseteq p_n$
  $op.type = \text{recon}$
  $op.phase = \text{put}$
  $cid = conf\text{-}id$
Effect:
  $recon\text{-}ip = \text{false}$
  $op.phase \leftarrow \text{idle}$

Fig. 7. Operation Manager OM_i Reconfiguration Transitions

ceives responses from every node in some put-quorum of the new configuration. OM_i may then broadcast a message to various focal-points, notifying them that the new configuration is established and that the reconfiguration is done.

4 Atomic Consistency

In this section, we discuss the proof that the GeoQuorums algorithm guarantees atomic consistency. For the complete proof, see the technical report [21]. The proof is divided into two parts. First, we show that each FPE acts like an atomic object with respect to put, get, confirm, and recon-done operations. Then we show that the OM guarantees atomic consistency.

Focal Point Emulator. The FPE uses the totally ordered LBCast service to implement a replicated state machine, which guarantees that the FPE implements an atomic object. If no new node joins a particular focal point after the beginning of the execution, it is easy to show that the FPE implements an atomic object: each request to the FPE is rebroadcast using the LBCast service; therefore every FPE receives requests in the same order. If the response for one operation precedes the request for a second operation, then clearly the request for the second comes after the request for the first in the LBCast total ordering. Therefore the second request will be processed after the first request.

The same conclusion holds when nodes join the focal point after the beginning of the execution. A joining node is sent a summary of all requests that occur prior to its beginning the join protocol, and receives from the LBCast service all requests for operations that occur after it begins the join protocol. Therefore, when the join protocol completes, the FPE has processed every request ordered

by the LBCast service prior to the completion of the join protocol. We conclude that the FPE implements an atomic object.

Operation Manager. The proof that the OM guarantees atomic consistency relies on establishing a partial order on read and write operations, based on the tag associated with each value. First, assume that all read operations complete in two phases (rather than being short-circuited by the *confirmed* flag). If no reconfiguration occurs, then it is easy to see that atomic consistency is guaranteed: assume operation π_1 completes before operation π_2 begins. First, assume that both use configuration c. Then π_1 accesses a put-quorum of c in its second phase, and π_2 accesses a get-quorum of c in its first phase. By the quorum intersection property, there is some focal point, h, that is in both quorums. Then focal point h first receives a message containing the request for π_1, and then sends a message in response to the request for π_2.

Next, assume that a reconfiguration occurs such that either π_1 or π_2 has the *data.recon-ip* flag set. The operation that has the flag set accesses quorums in both configurations c_1 and c_2 (or all configurations, if there are more than two), and therefore, as in the previous case, is guaranteed to contact a focal point, h, that is part of the quorum contacted by the other operation. Finally, assume that π_1 uses one configuration, say, c_1, and π_2 uses the other configuration, say, c_2, and that neither has set the *data.recon-ip* flag. Then at least one reconfiguration operation must begin during or after π_1 and complete before π_2, and we can show that this reconfiguration operation learns about π_1 in its first phase, and propagates information to π_2 in its second phase. (If there are more than two configurations, then the tag is conveyed from π_1 to π_2 because reconfiguration involves all existing configurations.)

Now we consider one-phase read operation. If a read operation terminates after one phase, then it has received a message that the associated tag has been confirmed. However, a tag is only confirmed when a prior operation has already completed the propagation of the tag.

Putting these pieces together, we obtain the following, which leads (by Lemma 13.16 in [26]) to the conclusion that atomic consistency is guaranteed:

Theorem 1. *If π_1 and π_2 are read/write operations, and π_1 completes before π_2 begins, then $tag(\pi_1) \leq tag(\pi_2)$. If π_2 is a write operation, then $tag(\pi_1) < tag(\pi_2)$.*

5 Performance Discussion

The performance of the GeoQuorums algorithm is directly dependent on the performance of the two communication services. Assume that every GeoCast message is delivered within time d_G, and every LBCast message is delivered within time d_{LB}; let $d = d_G + d_{LB}$. Then every read and write operation terminates within $8d$: each phase takes at most two round-trip messages. (An extra round of communication may be caused by the discovery during the first round that a reconfiguration is in progress.) The algorithm as specified also allows the implementation to trade-off message complexity and latency. In each phase, the node initiating the operation must contact a quorum of focal points. It can

accomplish this by sending one message to every focal point, thereby ensuring the fastest result, at the expense of a high message complexity. Alternatively, the node can send a message only to focal points in a single quorum. If not all responses are received (due, perhaps, to quorum members failing), the node can try another quorum, and continue until it receives a response from every member of some quorum. This leads to lower message complexity, but may take longer.

6 Conclusions and Future Work

We have presented a new approach, the GeoQuorums approach, to implementing algorithms in mobile, *ad hoc* networks. We have presented a geographic abstraction model, and an algorithm, the Focal Point Emulator, that implements it using mobile hosts. We have presented the Operation Manager, which uses the static model to implement an efficient, reconfigurable atomic read/write memory.

The GeoQuorums approach transforms a highly dynamic, *ad hoc* environment into a static setting. This approach should facilitate the adaptation of classical distributed algorithms to *ad hoc* networks. Unfortunately, the two components presented are tightly coupled: the implementation of the FPE is specific to the semantics of a reconfigurable atomic memory. We plan to further separate the two levels of the algorithm. This separation will allow the GeoQuorums approach to be applied to other challenging problems in mobile computing.

We also believe that our approach will be useful in studying hybrid networks, consisting of both mobile nodes and fixed infrastructure. In areas where there are non-mobile, fixed participants, simpler and more efficient versions of the FPE can be used. When nodes enter areas with no infrastructure, the more dynamic algorithm can seamlessly take over.

There are many open questions relating to the geographic abstraction. We have assumed a static definition of focal points and configurations, but it remains an open question to construct these in a distributed fashion, and to modify them dynamically. There are also questions related to the practical implementation of the model; we mention some ideas in Section 2, but open questions remain.

References

1. Dolev, S., Pradhan, D.K., Welch, J.L.: Modified tree structure for location management in mobile environments. Computer Communications: Special Issue on Mobile Computing **19** (1996) 335–345
2. Lamport, L.: On interprocess communication – parts I and II. Distributed Computing **1** (1986) 77–101
3. Herlihy, M.P., Wing, J.M.: Linearizability: A correctness condition for concurrent objects. ACM Trans. on Programming Languages and Systems **12** (1990) 463–492
4. Navas, J.C., Imielinski, T.: Geocast – geographic addressing and routing. In: ACM/IEEE Intl. Conference on Mobile Computing and Networking. (1997) 66–76
5. Camp, T., Liu, Y.: An adaptive mesh-based protocol for geocast routing. Journal of Parallel and Distributed Computing: Special Issue on Mobile Ad-hoc Networking and Computing (2002) 196–213

6. Gifford, D.K.: Weighted voting for replicated data. In: Proceedings of the seventh symposium on operating systems principles. (1979) 150–162
7. Thomas, R.H.: A majority consensus approach to concurrency control for multiple copy databases. Transactions on Database Systems **4** (1979) 180–209
8. Upfal, E., Wigderson, A.: How to share memory in a distributed system. Journal of the ACM **34** (1987) 116–127
9. Attiya, H., Bar-Noy, A., Dolev, D.: Sharing memory robustly in message-passing systems. Journal of the ACM **42** (1995) 124–142
10. Prisco, R.D., Fekete, A., Lynch, N.A., Shvartsman, A.A.: A dynamic primary configuration group communication service. In: Proceedings of the 13th International Symposium on Distributed Computing. (1999) 64–78
11. Lynch, N., Shvartsman., A.: RAMBO: A reconfigurable atomic memory service for dynamic networks. In: Proc. of the 16th Intl. Symp. on Distributed Computing. (2002) 173–190
12. Gilbert, S., Lynch, N., Shvartsman, A.: RAMBO II:: Rapidly reconfigurable atomic memory for dynamic networks. In: Proc. of the Intl. Conference on Dependable Systems and Networks. (2003) 259–269
13. Garcia-Molina, H., Barbara, D.: How to assign votes in a distributed system. Journal of the ACM **32** (1985) 841–860
14. Herlihy, M.: Dynamic quorum adjustment for partitioned data. Trans. on DB Systems **12** (1987) 170–194
15. El Abbadi, A., Skeen, D., Cristian, F.: An efficient fault-tolerant protocol for replicated data management. In: Proc. of the 4th Symp. on Principles of Databases, ACM Press (1985) 215–228
16. Dolev, D., Keidar, I., Lotem, E.Y.: Dynamic voting for consistent primary components. In: Proc. of the Sixteenth Annual ACM Symp. on Principles of Distributed Computing, ACM Press (1997) 63–71
17. Englert, B., Shvartsman, A.: Graceful quorum reconfiguration in a robust emulation of shared memory. In: Proc. of the International Conference on Distributed Computer Systems (ICDCS'2000). (2000) 454–463
18. Haas, Z.J., Liang, B.: Ad hoc mobile management with uniform quorum systems. IEEE/ACM Transactions on Networking **7** (1999) 228–240
19. Stojmenovic, I., Pena, P.E.V.: A scalable quorum based location update scheme for routing in ad hoc wireless networks. Technical Report TR-99-11, Computer Science, SITE, University of Ottawa (1999)
20. Karumanchi, G., Muralidharan, S., Prakash, R.: Information dissemination in partitionable mobile ad hoc networks. In: Proceedings of IEEE Symposium on Reliable Distributed Systems. (1999) 4–13
21. Dolev, S., Gilbert, S., Lynch, N.A., Shvartsman, A.A., Welch, J.L.: Geoquorums: Implementing atomic memory in ad hoc networks. Technical Report LCS-TR-900, MIT (2003)
22. Priyantha, N.B., Chakraborty, A., Balakrishnan, H.: The cricket location-support system. In: Proc. of the 6th ACM MOBICOM. (2000) 32–43
23. Ko, Y.B., Vaidya, N.: Geotora: A protocol for geocasting in mobile ad hoc networks. In: Proc. of the IEEE Intl. Conference on Network Protocols. (2000) 240–249
24. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. 2nd edn. Springer-Verlag (2000)
25. Dolev, S., Schiller, E., Welch, J.: Random walk for self-stabilizing group communication in ad-hoc networks. In: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems. (2002) 70–79
26. Lynch, N.: Distributed Algorithms. Morgan Kaufman (1996)