GeoQuorums: Implementing Atomic Memory in *Ad Hoc* Networks[*]

(Extended Abstract)

Shlomi Dolev[†] Seth Gilbert[‡] Nancy A. Lynch[‡] Alex A. Shvartsman[§] Jennifer L. Welch[¶]

Abstract

In this paper, we present a new approach for implementing atomic read/write shared memory in *ad hoc* networks. These networks are, by nature, highly dynamic, and it is therefore difficult to employ classical distributed algorithms. As a result, we divide the problem into two components. First, we define a geographic abstraction, associating virtual processes with physical regions, known as *focal points*, and we show how mobile hosts can implement this abstraction. Second, we present an atomic memory algorithm that depends on quorums of focal points to replicate the data, ensuring fault tolerance and consistency.

Our approach, then, is predicated on 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 landscape observation point, or a water resource in the desert. Mobile hosts that happen to populate a focal point participate in implementing shared atomic objects, storing their contents and supporting (remote) read and write operations.

The GeoQuorums algorithm defines certain intersecting sets of focal points, known as quorums. In order to complete a read or write operation, a mobile host contacts certain quorums of focal points. Using the capabilities of GPS, the new algorithm requires only a single round-trip communication phase to implement a write operation. Additionally, by maintaining information about the completion of specific reads and writes, the new algorithm can perform some read operations using only a single round-trip communication phase, as compared to previous algorithms that always require two phases. This reduction is especially effective in situations where read operations are frequent compared to write operations.

The observed frequency of read and write operations may lead to a change in the read and write policy: a different quorum system may lead to improved system performance. The algorithm allows for such changes to be made on the fly. Using the GeoQuorums approach and a limited number of possible quorum systems, we present a highly efficient configuration upgrade algorithm: reconfiguration always terminates rapidly (with no reliance on a consensus service) and has quite limited impact on concurrent read and write operations.

[*]This work was supported in part by NSF grant CCR-0098305 and NSF ITR Grant 0121277.

[†]Department of Computer Science, Ben-Gurion University, Beer-Sheva, 84105, Israel. Part of the work has been done during visits to MIT and Texas A&M. Partially supported by IBM faculty award, the Israeli ministry of defense, NSF, and the Israeli Ministry of Trade and Industry. e-mail: `dolev@cs.bgu.ac.il`.

[‡]MIT Laboratory of Computer Science, 545 Technology Square, NE43-365, Cambridge, MA 02139, USA. e-mail: `{sethg,lynch}@theory.lcs.mit.edu`. Partially supported by AFOSR Contract #F49620-00-1-0097, DARPA Contract #F33615-01-C-1896, NSF Grant 64961-CS, NTT Grant MIT9904-12

[§]Department of Computer Science and Engineering, 191 Auditorium Rd., Unit 3155, University of Connecticut, Storrs, CT 06269, USA and MIT Laboratory of Computer Science, 545 Technology Square, NE43-365, Cambridge, MA 02139, USA. e-mail: `alex@theory.lcs.mit.edu`. Partially supported by the NSF Grant 9988304 and by the NSF CAREER Award 9984774.

[¶]Department of Computer Science, Texas A&M University, College Station, TX 77843, e-mail: `welch@cs.tamu.edu`. Partially supported by NSF Grant 0098305 and Texas Advanced Research Program 000512-0091-2001.

1 Introduction

In this paper, we study the problem of designing algorithms for *ad hoc*, mobile networks. An *ad hoc* network uses no pre-existing infrastructure, unlike cellular networks that depend on fixed, wired base stations. Instead, the network is formed by the mobile nodes themselves, who 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 around the world, 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., [5]). 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 computing concepts to be applied in highly dynamic, *ad hoc* environments.

Providing atomic [18] (or linearizable [14]) read/write shared memory in *ad hoc* networks is a fundamental problem in distributed computing. Atomic memory provides 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. Similarly, a consistent 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 read/write memory in mobile, *ad hoc* networks.

The GeoQuorums Approach We divide the problem of implementing atomic read/write memory into two components. First, we define a static, abstract model that represents nodes in the network as well-defined geographic locales. We then show how to implement this model using mobile hosts. In this way, the dynamic nature of the *ad hoc* network is masked by a static model. Second, we present an algorithm for atomic memory that operates in the virtual 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.

The atomic memory algorithm is implemented on top of the geographic abstraction. Nodes implementing the atomic memory algorithm use a GeoCast service (as in [21, 2]) to communicate with the virtual processes, that is, with the focal points. In order to achieve fault tolerance and availability, the algorithm replicates the 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 [10, 25, 26, 1, 22]. The algorithm uses two sets of focal point quorums: (i) get-quorums, and (ii) put-quorums, with the property that every get-quorum intersects every put-quorum.

Our multi-writer/multi-reader algorithm takes advantage of 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 [26] or single writers [1]. Our algorithm also allows for some reads to be processed using a single phase: the Operation Manager flags the completion of a previous read or write to avoid using additional phases, and distributes this 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 [20, 11]. Reconfiguration does not significantly delay read or write operations, and, as no consensus service is required, reconfiguration terminates rapidly.

Another major benefit of using a quorum-based approach is fault tolerance. Quorum systems are inherently resilient to many types of failures. Any read or write operation is guaranteed to terminate whenever at least one get-quorum (for reads) and one put-quorum (for reads and writes) of the configurations contacted by the operation remain functional.

We formally specify all our algorithms in terms of Input/Output Automata [19]. The safety (atomicity) of the implementation is shown using assertional and partial-order techniques.

Overall, then, there are two primary contributions in this paper. First, we introduce the geographic abstraction model, which allows simple, static algorithms to be used effectively in highly dynamic environments. We provide an implementation, the Focal Point Emulator, of the model designed specifically for quorum based algorithms. Second, we present an atomic memory algorithm, the Operation Manager, that is coupled with the implementation of the geographic model. The new algorithm takes advantage of a real-time clock, provided by GPS, and improves on previous quorum based shared memory algorithms. It also provides a highly efficient reconfiguration service.

Other Approaches. Quorum systems are widely used to implement atomic memory in static distributed systems [10, 25, 26, 27, 1, 9, 13]. 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 [7, 15, 4, 13, 22] require the new configurations to be related to the old configurations, limiting their utility in *ad hoc* networks. Englert and Shvartsman [8] showed that using any two quorum systems concurrently preserves atomicity during more general reconfiguration. Recently, Lynch and Shvartsman introduced RAMBO [20] (extended in [11]), 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 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 [12] 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 [24] 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.* [16] 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. Finally, in Section 5, we conclude and present some areas for future research. The complete code for the algorithms and selected proofs are given in the (optional) Appendix.

2 System Model

In the first part of this section, we describe the basic environmental assumptions. In the second part, we present examples of real-world systems that support these assumptions.

Theoretical Model. Our world model consists of a bounded region of a two-dimensional plane, populated by mobile hosts. The mobile hosts may turn on, joining the system, or turn off, leaving the system. Since mobile hosts may fail at any time, we allow the hosts to leave the system without any protocol. The mobile hosts can move on any continuous path in the plane, where their maximum speed is bounded. 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 as to 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 non-intersecting regions in the plane, called *focal points*, such that (i) there are at least $n - f$ focal points (for some $f < n$), where at all times there is at least one mobile host in each focal point, 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 points.

Each mobile host also has a list of *configurations*. A configuration, c, consists of two sets of quorums: $\mathit{get\text{-}quorums}(c)$ and $\mathit{put\text{-}quorums}(c)$. Each quorum consists of a set of focal points, and they have the following intersection properties: if $G \in \mathit{get\text{-}quorums}(c)$ and $P \in \mathit{put\text{-}quorums}(c)$, then $G \cap P \neq \emptyset$. Additionally, for a given c, we assume that for any set of f focal points, F, there exist $G \in \mathit{get\text{-}quorums}(c)$ and $P \in \mathit{put\text{-}quorums}(c)$ 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. In this paper, we assume that there are only two configurations, c_1 and c_2.

Mobile hosts depend on two broadcast services: (i) Local-Broadcast, a local, atomic broadcast service, and (ii) GeoCast, a global delivery service. The Local-Broadcast service allows nodes within a focal point to communicate reliably. Each focal point is assumed to support a separate Local-Broadcast service: if we refer to focal point h, its broadcast service is referred to as $\mathit{Local\text{-}Broadcast}_h$. The Local-Broadcast service takes one parameter, a message, and delivers it to every node in the focal point. If mobile host i is in focal point h, and broadcasts a message m using $\mathit{Local\text{-}Broadcast}_h$ 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. The Global Positioning System (GPS) is perhaps the most common method. Others, like Cricket [23], are being developed to augment weaknesses in GPS, such as indoor operation. All the algorithms presented in this paper 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 [21] and later for *ad hoc* networks (e.g., [17, 2]).

We propose one set of configurations that may be particularly useful in actual implementations. We assume that accessing nearby focal points is faster than accessing distant focal points, and this set of configurations takes advantage of this locality principle. The focal points can be grouped into clusters, using some geographic technique [3]. Figure 1 illustrates the relationship among mobile hosts, focal points, and clusters. For configuration c_1, the $get\text{-}quorums$ are defined to be the clusters. The $put\text{-}quorums$ consist of every set containing exactly one node from each cluster. Configuration c_2 is defined in the opposite manner. If the clusters are relatively small and are well distributed in the plane, so that every mobile host is near to every focal point in some cluster, then configuration c_1 is quite efficient, if read operations are more common than write operations and if most read operations need only access the focal points of a single cluster. Similarly, in this case, configuration c_2 is quite efficient if write operations are more common than read operations. As we show later in this paper, our reconfiguration algorithm allows the system to safely switch between these two configurations.

Another implementation difficulty 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 [6]. 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.

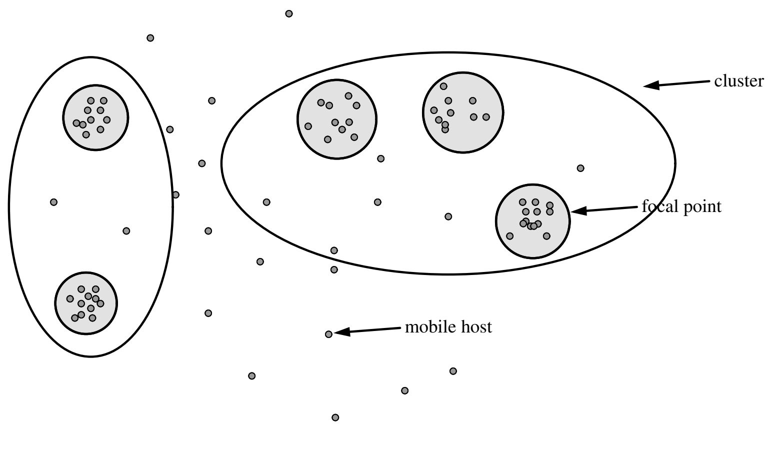

Figure 1: Clusters

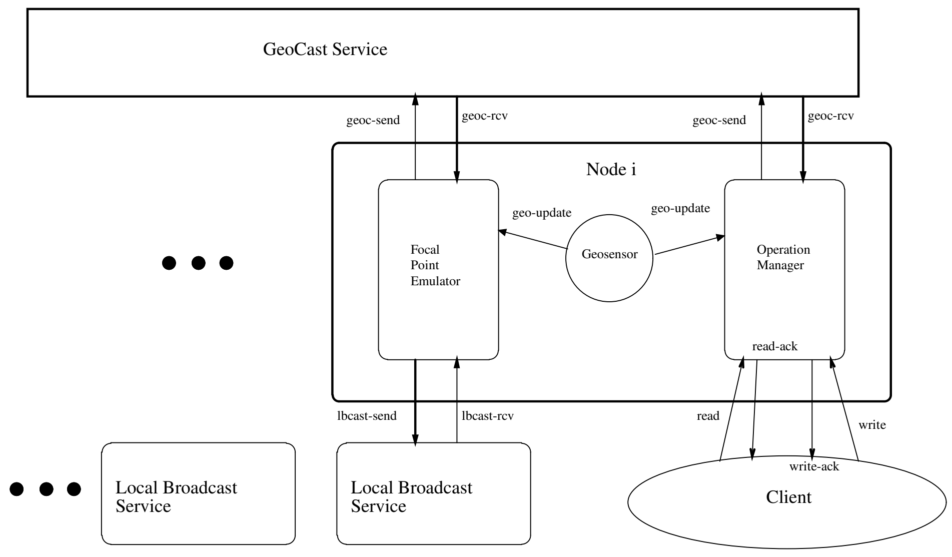

Figure 2: System Architecture

3 Focal Point Emulator and GeoQuorums Implementation

In this section we informally describe the algorithm used to implement read/write atomic memory in *ad hoc* networks. The algorithm consists of two components: the Focal Point Emulator, and the Operation Manager. Both of these algorithms are described in detail in Appendix A, where we present Input/Output Automata code for both automata.

We start with a high level description of the algorithms used in the system. Figure 2 describes the relationship of the different components of the mobile host program. Every mobile host may serve as a client that performs read and write memory accesses. For example, a client may request a read (the leftmost arrow from client to the operation manager) then a read procedure is invoked by the mobile host, sending GeoCast messages to, say, one focal point in each cluster ("geoc-send" arrow). The GeoCast message carries the current location of the sending mobile host, received from the geosensor (right "geo-update" arrow). Representatives of each focal point answer, using the position of the client sent in the GeoCast message, with the memory value and the time stamp of the last update ("geoc-rcv" arrow). Then, the mobile host uses the answers to compute the most updated value of the memory and responds to the client as part of the read-ack. A mobile host identifies the fact that it is in a focal point region by the information obtained from the geosensor (left "geo-update" arrow). Then the mobile host uses the local broadcast procedure to obtain a copy of the memory from other mobile hosts in the focal point region ("lbcast-send", "lbcast-rcv"). The mobile host is now ready to serve *put* and *get* operations to the memory arriving in messages of the GeoCast ("geoc-send" and "geoc-rcv" arrows). We next 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.

The FPE maintains a $data$ record that represents the state being replicated at every mobile host in the focal point. The FPE receives GeoCast messages updating the value of the atomic data object, which it stores in $data.value$. Each update is accompanied by a unique tag from a totally ordered set, that is stored in $data.tag$. Occasionally the FPE is notified that a tag is confirmed; $data.confirmed$ tracks the set of confirmed tags. Many requests to the FPE contain the ID of a configuration; $data.conf\text{-}id$ stores the largest known configuration ID. Occasionally the FPE is notified that a configuration is completely installed; $data.recon\text{-}ip$ is a flag that indicates whether a reconfiguration is in progress.

The FPE receives various messages from the GeoCast service, sent by a mobile host. The incoming message is immediately rebroadcast, using the Local Broadcast service. The FPE takes no other action in response to GeoCast messages.

The FPE also receives messages from the Local Broadcast service. Each FPE automaton can be either idle, joining or active. If it is not idle, then it will process the message and update its local state. Even if a node is in the process of joining, it updates its local state to maintain consistency. If the node is active (and thus no joining is in progress), then the FPE enqueues a response, if required. Whenever a response is sent through the GeoCast service, an ack message is sent using the Local Broadcast service. If a node has received an ack message for a given request, it does not need to send a response. Similarly, if a node notices that it has already handled a given request, then it does not need to send a response. Finally, if any node notices that a new configuration is being used, it sets a flag to remember that a reconfiguration is in progress.

We now discuss the various messages received by node i from the Local Broadcast service. The first four are messages rebroadcast from the Local Broadcast service. (i) If node i receives a *get* message and if certain criteria are met, then node i sends a response via the GeoCast service, containing its current copy of tag, value, and confirmed (ii) If node i receives a *put* message, then node i updates its local copy of tag, value and confirmed using the data in the message. If certain criteria are met, then node i sends a response using the GeoCast service, indicating that the update is complete. (iii) If node 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 Focal Point Emulator 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 Local Broadcast 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 certain criteria are met, node i sends a response using the Local Broadcast; this response includes information about the current value of the shared object, the tag, and whether or not the value is confirmed. As soon as the initiator of the join protocol receives any response, it updates its current local copy of the value, the tag, and the confirmed flag with the information in the response message, and then becomes active.

3.2 Operation Manager

The Operation Manager uses the GeoCast service to communicate with focal points, sending get, put, and confirm messages to various focal points, and receiving appropriate responses. The Operation Manager uses the focal points as replicas, guaranteeing both atomicity and fault tolerance. For each phase of each operation, the automaton will contact a quorum of focal points. The quorum intersection property will ensure consistency.

Read/Write Operations When node i receives a write request, it first examines its clock to choose a unique tag for the operation. Assume that node i believes the current configuration is c. Node i then uses the GeoCast service to send the new tag and value to be written to a number of focal points. If no reconfiguration is in progress, and if all responses indicate that c is the only known configuration, then the operation terminates when node 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 to configuration c', or that c'

is known anywhere as the newer current configuration, then node i must wait until it also receives responses from each focal point in some $P' \in \textit{put-quorums}(c')$. After the operation is complete, node i can optionally notify focal points that the specified tag has been *confirmed*, indicating that the operation is complete.

When node i receives a read request, it sends out messages to a number of focal points. As before, if no focal point indicates that a reconfiguration is in progress, and no focal point indicates the existence of a newer configuration, then the first phase terminates when i receives a response from each focal point in some $G \in \textit{get-quorums}(c)$. Otherwise, the phase only completes when it also hears from each focal point in some $G' \in \textit{get-quorums}(c')$, where c' is the other configuration. At this point, node 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, node 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 notifying a put-quorum that the operation has completed, the algorithm allows later operations to discover that a second phase is unnecessary.

Reconfiguration The reconfiguration algorithm is, in some manners, a variant of the reconfiguration mechanism presented in the RAMBO II algorithm [11]: 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. For the rest of this paper, we assume that there are exactly two configurations, named c_1 and c_2. 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. First, node i chooses a new, unique, configuration identifier, by examining the local clock, its node-id, and the name of the desired configuration. Then node i sets a flag, indicating that a reconfiguration is in progress. At this point, the first phase of the reconfiguration begins, sending messages to a number of focal points. The first phase terminates when it 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 (iii) put-quorum of c_2. Then the second phase begins, again sending out messages to focal points. It terminates when i receives responses from every node in some put-quorum of the new configuration. Node 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 Proof of Atomic Consistency

In this section, we discuss the proof that the algorithm presented guarantees atomic consistency. For the details of the proof, see Appendix C. The proof is divided into two parts. First, we show that each focal point, running the Focal Point Emulator, acts like a single, atomic object. Then we show that the Operation Manager correctly uses these focal point objects to provide atomic consistency.

Focal Point Emulator The first fact to note about the FPE is that, if nodes never join or leave a focal point, it is clear that it implements an atomic object. Each request to the focal point is rebroadcast using the $\textit{lbcast}$ service, which assigns a total ordering to all requests. If every node in the focal point begins in the same initial state, and every node in the focal point handles every request, and the requests are handled in the same order (that determined by the $\textit{lbcast}$ service), then if two mobile hosts i, j have both just processed message m, then they are both in the same state and send the same response. We claim the same result for the more general case when nodes frequently join and leave the focal point:

Lemma 4.1 *Assume that at some point in the execution, mobile host i has an item in its outgoing GeoCast queue of the following form:* $\langle op_1, oid, payload_1\rangle$. *Further, assume that at some (other) point in the execu-*

tion mobile host j has an item in its outgoing GeoCast queue of the following form: $\langle op_2, oid, payload_2\rangle$. Then, $op_1 = op_2$, and $payload_1 = payload_2$.

This shows that if two different nodes in a single focal point both GeoCast a response to the same request (identified by oid), then they always send the same response.

Next, we claim that put and get operations act atomically. For example, if a put request stores some value at the focal point, a later get request returns that value, or the value of a more recent put request. Similarly, the configuration-ids returned in response to requests are non-decreasing.

Lemma 4.2 *Assume that i and j are two mobile hosts; all described actions occur in focal point h. Assume that i GeoCasts a response to an operation, π, where π is either a* put *or a* get *request. Later, j GeoCasts a response to a* get *request. Then the tag sent by j is no smaller than the tag sent or received for π, and the configuration-id sent j is no smaller than the configuration-id sent or received for π.*

We also show that similar results hold for the *confirmed* flag and the rconfiguration flag: if a focal point sends a response that a certain tag is confirmed, then it must previously have received a request indicating that the tag was confirmed; if a focal point sends a response indicating that a certain reconfiguration is complete, then it must previously have received a request indicating that the reconfiguration was complete.

Operation Manager Once it has been shown that the focal point emulates an atomic object, we can show that the operation manager preserves atomicity. The complete proofs are given in Appendix C. The proof relies on establishing a partial order on read and write operations based on tags using the approach of Lemma 13.16 in [19].

We first show that if there is no reconfiguration, and all read operations complete in two phases (rather than being short-circuited by the *confirmed* flag) then atomic consistency is guaranteed. Assume you have two operations, π_1 and π_2, and the former completes before the latter begins. Assume that both use configuration c. Then π_1 accesses a put-quorum of c, and π_2 accesses a get-quorum of c. 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 π_1, and then sends a message in response to π_2. By Lemma 4.2, the second message contains information as least as recent as that received in the first message. This shows the following:

Lemma 4.3 *If π_1 is a two-phase read or write operation, and π_2 is a read or write operation, and π_1 and π_2 both use the same configuration, and if π_1 completes before π_2 begins, then the tag associated with π_1 is no larger than the tag associated with π_2, and if π_2 is a write operation, then the tag associated with π_1 is strictly less than the tag associated with π_2.*

The second key aspect of the proof is showing that the reconfiguration mechanism does not violate atomicity. Again consider two operation π_1 and π_2. If either π_1 or π_2 has the flag set indicating that a reconfiguration is in progress, it is again straightforward to see that consistency is maintained: the operation that has the flag set accesses quorums in both configurations c_1 and c_2, and therefore, as in the previous case, is guaranteed to contact a quorum that intersects with a quorum accessed by the other operation.

Assume instead that each operation accesses only a single configuration (as the reconfiguration in progress flag is set otherwise). Assume that π_1 uses configuration c, and π_2 uses configuration c'. We show that c' must be at least as recent a configuration as c. Then we know that the reconfiguration that establishes c' completes before π_2. We then show that this reconfiguration propagates information from π_1, discovered in the first phase, to π_2 in the second phase. This shows that:

Lemma 4.4 *If π_1 is a two-phase read or write operation, and π_2 is a read or write operation, and π_1 uses a smaller configuration than π_2, and if π_1 completes before π_2 begins, then the tag associated with π_1 is no larger than the tag associated with π_2, and if π_2 is a write operation, then the tag associated with π_1 is strictly less then the tag associated with π_2.*

The last aspect to consider is one-phase read operations. Consider a read operation that stops after its first phase as a result of the *confirmed* flag. We then show that there must be an operation with the same associated tag and value that completes before the read operation begins, and that accesses a *put-quorum*. This then leads to the conclusion that atomicity is preserved. Putting these pieces together, we show the following, which leads (by Lemma 13.16 in [19]) to the conclusion that atomic consistency is guaranteed:

Lemma 4.5 *If π_1 and π_2 are read or write operations, and π_1 completes before π_2 begins, then the tag associated with π_2 is no smaller than the tag associated with π_1. If π_2 is also a write operation, then the tag associated with π_1 is strictly less than the tag associated with π_2.*

5 Conclusions and Future Work

In this paper, we have presented a new approach to implementing algorithms in highly dynamic, mobile networks. To solve the problem of atomic memory, we have presented a geographic abstraction model, and an algorithm, the Focal Point Emulator, that implements it using mobile hosts. We have also presented a client algorithm, the Operation Manager, that takes advantage of the static abstraction to implement an efficient, reconfigurable atomic read/write memory.

In this paper, the two algorithms presented are tightly coupled: the Operation Manager depends on the Focal Point Emulator's awareness of the semantics of reconfigurable atomic memory. We hope in the future to develop a more abstract model for the Focal Point Emulator. It should be possible to specify the desired behavior as an atomic data object, and the focal point should guarantee that the specification is met. This would simplify the development and proof of algorithms in the GeoQuorums model.

It seems likely that many algorithmic problems should be amenable to the approach in this paper. The geographic abstraction transforms the highly dynamic, *ad hoc* environment into a relatively static model. This transformation should facilitate the use in *ad hoc* networks of many classical distributed algorithms that are otherwise difficult to implement in highly dynamic environments.

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 best way to implement the geographic abstraction. In this paper, we assumed a static definition of focal points and configurations. Yet it might be useful to construct these entities in a distributed fashion, and to modify them as the algorithm runs. There are also questions as to how to implement the various components of this algorithm. We mentioned some ideas in Section 2 as to what a practical implementation might look like, but many other options are possible.

References

[1] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 42(1):124–142, 1995.

[2] T. Camp and Y. Liu. An adaptive mesh-based protocol for geocast routing. *Journal of Parallel and Distributed Computing: Special Issue on Mobile Ad-hoc Networking and Computing*, 2002.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[4] Danny Dolev, Idit Keidar, and Esti Yeger Lotem. Dynamic voting for consistent primary components. In *Proc. of the Sixteenth Annual ACM Symp. on Principles of Distributed Computing*, pages 63–71. ACM Press, 1997.

[5] S. Dolev, D. K. Pradhan, and J. L. Welch. Modified tree structure for location management in mobile environments. *Computer Communications: Special Issue on Mobile Computing*, 19(4):335–345, April 1996.

[6] S. Dolev, E. Schiller, and J. Welch. Random walk for self-stabilizing group communication in ad-hoc networks. In *To appear in Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, 2002.

[7] El Abbadi, Skeen, and Cristian. An efficient fault-tolerant protocol for replicated data management. In *Proceedings of the 4th Annual ACM Symposium on Principles of Databases*, pages 215–228, 1985.

[8] B. Englert and A.A. Shvartsman. Graceful quorum reconfiguration in a robust emulation of shared memory. In *Proc. of the International Conference on Distributed Computer Systems (ICDCS'2000)*, pages 454–463, 2000.

[9] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, 1985.

[10] David K. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh symposium on Operating systems principles*, pages 150–162, 1979.

[11] Seth Gilbert, Nancy Lynch, and Alex Shvartsman. RAMBO II:: Rapidly reconfigurable atomic memory for dynamic networks. In *Proc. of the International Conference on Dependable Systems and Networks*, June 2003.

[12] Z. J. Haas and B. Liang. Ad hoc mobile management with uniform quorum systems. *IEEE/ACM Transactions on Networking*, 7(2), April 1999.

[13] Maurice Herlihy. Dynamic quorum adjustment for partitioned data. *Trans. on DB Systems*, 12(2):170–194, 1987.

[14] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, July 1990.

[15] S. Jajodia and David Mutchler. Dynamic voting algorithms for maintaining the consistency of a replicated database. *Transactions on Database Systems*, 15(2):230–280, 1990.

[16] G. Karumanchi, S. Muralidharan, and R. Parkash. Information dissemination in partitionable mobile ad hoc networks. In *Proceedings of IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 4–13, 1999.

[17] Young Bae Ko and Nitin Vaidya. Geotora: A protocol for geocasting in mobile ad hoc networks. In *Proc. of the IEEE International Conference on Network Protocols*, pages 240–249, November 2000.

[18] Leslie Lamport. On interprocess communication – parts I and II. *Distributed Computing*, 1(2):77–101, April 1986.

[19] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.

[20] Nancy Lynch and Alex Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In *Proceedings of the 16th Intl. Symposium on Distributed Computing*, pages 173–190, 2002.

[21] J. C. Navas and T. Imielinski. Geocast – geographic addressing and routing. In *ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, 1997.

[22] Roberto De Prisco, Alan Fekete, Nancy A. Lynch, and Alexander A. Shvartsman. A dynamic primary configuration group communication service. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 64–78, September 1999.

[23] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *Proc. of the 6th ACM MOBICOM*, August 2000.

[24] I. Stojmenovic and P. E. V. Pena. 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, December 1999.

[25] Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *Transactions on Database Systems*, 4(2):180–209, 1979.

[26] Eli Upfal and Avi Wigderson. How to share memory in a distributed system. *Journal of the ACM*, 34(1):116–127, 1987.

[27] P.M.B. Vitányi and B. Awerbuch. Atomic shared register access by asynchronous hardware. In *Proceedings 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 233–243, New York, 1986. IEEE.

Appendices

A Automata Specifications: Focal Point Emulator

Domains:

I, set of mobile hosts identifiers
V, set of values $\cup \perp$
B, set of boolean values $\cup \perp$
C, set of clock times
T, set of tags $C \times I \cup \{\perp\}$
O, set of operation ids $C \times I$
L, set of locations, presumably $\mathcal{R} \times \mathcal{R} \cup \perp$
D, a set of destinations, $I \cup \{\textsf{focal-point}\} \times L$
FP, a set of focal points,
  presumably $L \times \mathcal{R} \cup \perp$
N, set of configuration-names $\{0, 1\}$
CID, set of configuration-ids $C \times I \times N$

Figure 3: Domains

FPE_i**: State Components**

$status \in \{\textsf{idle}, \textsf{joining}, \textsf{active}\}$, initially $\begin{cases} \textsf{idle} & \text{if } i \text{ is not initially in any focal point} \\ \textsf{active} & \text{if } i \text{ is initially in some focal point} \end{cases}$
$\mathit{focalpoint\text{-}id} \in FP$, initially $\perp$
$\mathit{join\text{-}oid} \in O$, initially 0
$\mathit{complete\text{-}ops}$, a subset of O, initially $\emptyset$
$data$, a record with fields:
  $value \in V$, initially v_0
  $tag \in T$, initially $\perp$
  $\mathit{confirmed}$, a subset of T, initially $\emptyset$
  $\mathit{conf\text{-}id} \in CID$, initially $\langle 0, 0, 0\rangle$
  $\mathit{recon\text{-}ip} \in B$, initially $\textsf{false}$
$queues$, a record with fields:
  $geocast$, a queue of $\langle op, payload, src, dest\rangle$, initially $\emptyset$
  $lbcast$, a queue of $\langle op, payload, \mathit{op\text{-}src}\rangle$, initially $\emptyset$
$global$, a record with fields:
  $\mathit{fp\text{-}map}$, a set of FP, initially $\emptyset$
  $clock \in C$, initially 0
  $location \in L$, initially i's initial location

Figure 4: Focal Point Emulator State

FPE_i: **Signature:**

Input:

```
lbcast-rcv($message, payload, op\text{-}src$)$_{h,i}$,
  $i \in I$,
  $h \in FP$,
  $message \in$ {get, put, ack, confirm, recon-done,
                 join-req, join-req-ack}
  $payload \in \{(O \times CID)$,
                 $(O \times T \times V \times B \times CID)$,
                 $(O)$,
                 $(O \times T)$,
                 $(O \times CID)$,
                 $(O \times V \times T \times \mathcal{P}(T) \times CID \times B \times \mathcal{P}(O))\}$,
  $op\text{-}src \in D$

geoc-rcv($message, payload, src, dest$)$_i$,
  $i \in I$,
  $message \in$ {get, put, confirm, recon-done}
  $payload \in \{(O \times CID)$,
                 $(O \times T \times V \times B \times CID)$,
                 $(O \times T)$,
                 $(O \times CID)\}$,
  $src, dest \in D$

geo-update($current\text{-}loc, new\text{-}time$)$_i$,
  $i \in I$
  $current\text{-}loc \in L$
  $new\text{-}time \in C$
```

Output:

```
lbcast-send($message, payload, op\text{-}src$)$_{h,i}$,
  $i \in I$,
  $h \in FP$,
  $message \in$ {get, put, ack, confirm, recon-done,
                 join-req, join-req-ack}
  $payload \in \{(O \times CID)$,
                 $(O \times T \times V \times B \times CID)$,
                 $(O)$,
                 $(O \times T)$,
                 $(O \times CID)$,
                 $(O \times T \times V \times \mathcal{P}(T) \times CID \times B \times \mathcal{P}(O))\}$,
  $op\text{-}src \in D$

geoc-send($message, payload, src, dest$)$_i$,
  $i \in I$
  $message \in$ {get-ack, put-ack},
  $payload \in \{(O \times T \times V \times B \times CID \times B)$,
                 $(O \times CID \times B)$,
  $src, dest \in D$
```

Internal:

```
join()$_i$
  $i \in I$

leave()$_i$
  $i \in I$
```

Figure 5: Focal Point Emulator Signature

FPE_i: **Transitions I**

```
Input lbcast-rcv(get, $oid$, $cid$, $op\text{-}src$)$_{h,i}$
Effect:
  if ($h = lookup(global.location, global.fp\text{-}map)$) then
    if $cid > data.conf\text{-}id$ then
      $data.conf\text{-}id \leftarrow cid$
      $data.recon\text{-}ip \leftarrow$ true
    if $status =$ active then
      if $oid \notin complete\text{-}ops$ then
        $conf \leftarrow data.tag \in data.confirmed$
        $src \leftarrow \langle$focal-point, $focalpoint\text{-}id\rangle$
        $dest \leftarrow op\text{-}src$
        Enqueue($queues.geocast$,
              $\langle$get-ack, $oid$, $data.tag$, $data.value$, $conf$,
               $data.conf\text{-}id$, $data.recon\text{-}ip$, $src$, $dest\rangle$)
    $complete\text{-}ops \leftarrow complete\text{-}ops \cup \{oid\}$

Input lbcast-rcv(put, $oid$, $cid$, $op\text{-}src$)$_{h,i}$
Effect:
  if ($h = lookup(global.location, global.fp\text{-}map)$) then
    if $cid > data.conf\text{-}id$ then
      $data.conf\text{-}id \leftarrow cid$
      $data.recon\text{-}ip \leftarrow$ true
    if $t > data.tag$ then
      $data.value \leftarrow v$
      $data.tag \leftarrow t$
    if $status =$ active then
      if $oid \notin complete\text{-}ops$ then
        $src = \langle$focal-point, $focalpoint\text{-}id\rangle$
        $dest = op\text{-}src$
        Enqueue($queues.geocast$,
              $\langle$put-ack, $oid$, $data.conf\text{-}id$, $data.recon\text{-}ip$, $src$, $dest\rangle$)
    $complete\text{-}ops \leftarrow complete\text{-}ops \cup \{oid\}$

Input lbcast-rcv(ack, $oid$)$_{h,i}$
Effect:
  if ($h = lookup(global.location, global.fp\text{-}map)$) then
    $\forall x = \langle *, oid, *, *, * \rangle \in queues.geocast$
      Remove $x$ from $queues.geoccast$
    $\forall x = \langle *, oid, *, * \rangle \in queues.lbcast$
      Remove $x$ from $queues.lbcast$

Input geoc-rcv($\langle op, payload, src, dest\rangle$)$_i$
Effect:
  Enqueue($queues.lbcast$, $\langle op, payload, src\rangle$)

Output geoc-send($op, payload, src, dest$)
Precondition:
  $\langle oid, * \rangle = payload$
  $\langle op, payload, src, dest\rangle = queues.lbcast.head$
Effect:
  $queues.geocast \leftarrow queues.geocast.tail$
  if ($op =$ get || $op =$ put) then
    Enqueue($queues.lbcast$, $\langle$ack, $oid$, $\perp\rangle$)

Output lbcast-send($op, payload$)$_{h,i}$
Precondition:
  $h = lookup(global.location, global.fp\text{-}map)$
  $\langle op, payload\rangle = queues.lbcast.head$
Effect:
  $queues.lbcast \leftarrow queues.lbcast.tail$

Input lbcast-rcv(confirm, $oid$, $t$, $op\text{-}src$)$_{h,i}$
Effect:
  if ($h = lookup(global.location, global.fp\text{-}map)$) then
    $data.confirmed \leftarrow data.confirmed \cup t$
    $complete\text{-}ops \leftarrow complete\text{-}ops \cup \{oid\}$

Input lbcast-rcv(recon-done, $oid$, $cid$, $op\text{-}src$)$_{h,i}$
Effect:
  if ($h = lookup(global.location, global.fp\text{-}map)$) then
    if $cid = data.conf\text{-}id$ then
      $data.recon\text{-}ip \leftarrow$ false
    $complete\text{-}ops \leftarrow complete\text{-}ops \cup \{oid\}$

Input geo-update($new\text{-}loc$, $new\text{-}time$)$_i$
Effect:
  $global.location \leftarrow current\text{-}loc$
  $global.clock \leftarrow new\text{-}time$
```

Figure 6: Focal Point Emulator send/receive Transitions

FPE_i**: Transitions II**

```
Internal $\mathsf{join}()_i$
Precondition:
    $\mathsf{lookup}(global.location, global.fp\text{-}map) \neq \bot$
    $focalpoint\text{-}id = \bot$
    $status = \mathsf{idle}$
Effect:
    $focalpoint\text{-}id \leftarrow \mathsf{lookup}(global.location, global.fp\text{-}map)$
    $status = \mathsf{joining}$
    $join\text{-}oid = \langle global.clock, i \rangle$
    $data.conf\text{-}id \leftarrow \langle 0, 0, 0 \rangle$
    Enqueue($queues.lbcast$, $\langle \mathsf{join\text{-}req}, join\text{-}oid, \bot \rangle$)

Input $\mathsf{lbcast\text{-}rcv}(\mathsf{join\text{-}req}, oid, op\text{-}src)_{h,i}$
Effect:
    if $(h = lookup(global.location, global.fp\text{-}map))$ then
      if $status = \mathsf{active}$ then
        if $oid \notin complete\text{-}ops$ then
          Enqueue($lbcast\text{-}queue$, $\langle \mathsf{join\text{-}req\text{-}ack}, oid, data, complete\text{-}ops, \bot \rangle$)
      $complete\text{-}ops \leftarrow complete\text{-}ops \cup \{oid\}$

Internal $\mathsf{leave}()_i$
Precondition:
    $focalpoint\text{-}id \neq \mathsf{lookup}(global.location, global.fp\text{-}map_i)$
Effect:
    $data.value \leftarrow \bot$
    $data.tag \leftarrow \bot$
    $data.confirmed \leftarrow \emptyset$
    $focalpoint\text{-}id \leftarrow \bot$
    $status = \mathsf{idle}$

Input $\mathsf{lbcast\text{-}rcv}(\mathsf{join\text{-}req\text{-}ack}, oid, new\text{-}data, new\text{-}complete\text{-}ops)_{h,i}$
Effect:
    if $(h = lookup(global.location, global.fp\text{-}map))$ then
      if $status = \mathsf{joining}$ then
        if $join\text{-}oid = oid$ then
          if $new\text{-}data.tag > data.tag$ then
            $data.tag \leftarrow new\text{-}data.tag$
            $data.value \leftarrow new\text{-}data.value$
          $data.confirmed \leftarrow data.confirmed \cup new\text{-}data.confirmed$
          if $(data.conf\text{-}id \leq new\text{-}data.conf\text{-}id)$ then
            $data.conf\text{-}id \leftarrow new\text{-}data.conf\text{-}id$
            $data.recon\text{-}ip \leftarrow new\text{-}data.recon\text{-}ip$
          $complete\text{-}ops \leftarrow complete\text{-}ops \cup new\text{-}complete\text{-}ops$
          $status = \mathsf{active}$
          $join\text{-}oid = \bot$
      $\forall x = \langle *, oid, *, *, * \rangle \in queues.geocast$
        Remove $x$ from $queues.geocast$
      $\forall x = \langle *, oid, *, * \rangle \in queues.lbcast$
        Remove $x$ from $queues.lbcast$
```

Figure 7: Focal Point Emulator join/leave Transitions

B Automata Specifications: Operation Manager

OM_i: **State Components**

```
$confirmed$, a subset of $T$, initially $\emptyset$
$conf\text{-}id \in CID$, initially $\langle 0, 0, 0\rangle$
$recon\text{-}ip \in B$, initially false
$op$, a record with fields:
    $type \in \{\mathsf{read}, \mathsf{write}, \mathsf{recon}\}$
    $phase \in \{\mathsf{idle}, \mathsf{get}, \mathsf{put}\}$, initially idle
    $tag \in T$, initially $\perp$
    $value \in V$, initially $\perp$
    $recon\text{-}in\text{-}progress \in B$, initially false
    $oid \in O$, initially 0
    $acc$, a finite subset of $I$, initially $\emptyset$
    $loc \in L$, initially $\perp$
$global$, a record with fields:
    $location \in L$, initially $\perp$
    $clock \in C$, initially 0
    $fp\text{-}map$, a subset of $FP$, initially a set of focal point definitions
$G_1 \subseteq \mathcal{P}(FP)$, initially the set of get-quorums for configuration 1
$P_1 \subseteq \mathcal{P}(FP)$, initially the set of put-quorums for configuration 1
$G_2 \subseteq \mathcal{P}(FP)$, initially the set of get-quorums for configuration 2
$P_2 \subseteq \mathcal{P}(FP)$, initially the set of put-quorums for configuration 2
```

OM_i: **Signature**

```
Input:

  read()$_i$
    $i \in I$
  write($v$)$_i$,
    $i \in I$,
    $v \in V$
  recon($conf\text{-}name$)$_i$
    $i \in I$,
    $conf\text{-}name \in N$

Output:

  read-ack($v$)$_i$
    $i \in I$,
    $v \in V$
  write-ack()$_i$
    $i \in I$
  recon-ack()$_i$
    $i \in I$

Internal:

  read-2()$_i$
    $i \in I$
  recon-2()
    $i \in I$
  confirm()$_i$
    $i \in I$

Input:

  geoc-rcv($op, payload, src, dest$)$_i$
    $op \in \{\mathsf{get\text{-}ack}, \mathsf{put\text{-}ack}\}$,
    $payload \in \{(O \times T \times V \times B \times CID \times B)\}$,
                $(O \times CID \times B)\}$,
    $src, dest \in D$

Output:

  geoc-send($op, payload, src, dest$)$_i$
    $op \in \{\mathsf{get}, \mathsf{put}, \mathsf{confirm}, \mathsf{recon\text{-}done}\}$
    $payload \in \{(O \times CID)$,
                $(O \times T \times V \times B \times CID)$,
                $(O \times T)$,
                $(O \times CID)$,
    $src, dest \in D$
```

Figure 8: Operation Manager State and Signature

OM_i: **Transitions I**

```
Input read()_i
Effect:
    $op.type \leftarrow$ read
    $op.phase \leftarrow$ get
    $op.tag \leftarrow \bot$
    $op.value \leftarrow \bot$
    $op.recon\text{-}in\text{-}progress \leftarrow recon\text{-}ip$
    $op.oid \leftarrow \langle global.clock, i\rangle$
    $op.acc \leftarrow \emptyset$
    $op.loc \leftarrow global.location$

Input write($v$)_i
Effect:
    $op.type \leftarrow$ write
    $op.phase \leftarrow$ put
    $op.tag \leftarrow \langle global.clock, i\rangle$
    $op.value \leftarrow v$
    $op.recon\text{-}in\text{-}progress \leftarrow recon\text{-}ip$
    $op.oid \leftarrow \langle global.clock, i\rangle$
    $op.acc \leftarrow \emptyset$
    $op.loc \leftarrow global.location$

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

```
Input geoc-rcv(get-ack, $oid$, $tag$, $val$, $conf$, $cid$, $rec\text{-}ip$, $src$, $dest$)_i
Effect:
    if $op.oid = oid$ then
      if $tag > op.tag$ 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
      $conf\text{-}id \leftarrow cid$
      $op.recon\text{-}in\text{-}progress \leftarrow$ 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
      $confirmed \leftarrow confirmed \cup \{tag\}$

Input geoc-rcv(put-ack, $oid$, $cid$, $rec\text{-}ip$, $src$, $dest$)_i
Effect:
    if $op.oid = oid$ then
      $acc \leftarrow acc \cup \{lookup(src.loc, global.fp\text{-}map)\}$
    if $cid > conf\text{-}id$ then
      $conf\text{-}id \leftarrow cid$
      $op.recon\text{-}in\text{-}progress \leftarrow$ true
      $recon\text{-}ip \leftarrow$ true
    else if $cid = conf\text{-}id$ then
      if $r\text{-}ip =$ 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
      $message = op.phase$ || $message \in$ {confirm, recon-done}
    else
      $message \in$ {confirm, recon-done}
    $cf = op.tag \in confirmed$
    if ($op.phase =$ get) then
      $payload = \langle op.oid, conf\text{-}id\rangle$
    if ($op.phase =$ put) then
      $payload = \langle op.oid, op.tag, op.value, cf, conf\text{-}id\rangle$
    if ($op.phase =$ confirm) then
      $op.tag \in confirmed$
      $payload = \langle op.oid, op.tag\rangle$
    if ($op.phase =$ recon-done) then
      $recon\text{-}ip =$ false
      $payload = \langle op.oid, conf\text{-}id\rangle$
    $src = \langle i, global.location\rangle$
    $fp\text{-}name \in FP$
    $dest = \langle$focal-point, $fp\text{-}name\rangle$
Effect:
    None
```

Figure 9: Operation Manager Transitions

OM_i: **Transitions II**

```
Output read-ack$(v)_i$
Precondition:
    $conf\text{-}id = \langle c, p, n\rangle$
    if $op.recon\text{-}in\text{-}progress$ 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 read-ack$(v)_i$
Precondition:
    $conf\text{-}id = \langle c, p, n\rangle$
    if $op.recon\text{-}in\text{-}progress$ 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:
    $op.phase \leftarrow \mathsf{idle}$

Output write-ack$()_i$
Precondition:
    $conf\text{-}id = \langle c, p, n\rangle$
    if $op.recon\text{-}in\text{-}progress$ 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:
    $op.phase \leftarrow \mathsf{idle}$
    $confirmed \leftarrow confirmed \cup \{op.tag\}$
```

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

Internal recon-upgrade$(cid)_i$
Precondition:
    $recon\text{-}ip = \mathsf{true}$
    $op.phase = \mathsf{idle}$
    $cid = conf\text{-}id$
Effect:
    $op.type \leftarrow \mathsf{recon}$
    $op.phase \leftarrow \mathsf{get}$
    $op.tag \leftarrow \bot$
    $op.value \leftarrow \bot$
    $op.recon\text{-}in\text{-}progress \leftarrow \mathsf{true}$
    $op.oid \leftarrow \langle global.clock, i\rangle$
    $op.acc \leftarrow \emptyset$
    $op.loc \leftarrow global.location$

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:
      $acc \supseteq g_0 \cup g_1 \cup p_0 \cup p_1$
    $op.type = \mathsf{recon}$
    $op.phase = \mathsf{get}$
    $cid = conf\text{-}id$

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

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

Figure 10: Operation Manager Transitions (continued)

C Safety Guarantees: Atomic Memory

In this section we prove that the GeoQuorums algorithm implements a distributed atomic memory.

C.1 Focal Point Emulator

There are two major theorems to prove about the Focal Point Emulator. The first theorem shows that if two nodes, i and j, both respond to a request, then they both send the same response. First we need some lemmas showing how nodes respond to messages, and how nodes join the system.

We prove results about the FPE for some fixed focal point, h. Throughout this discussion, assume that α is an execution. Also, assume that M is a finite prefix, $\{m_0, \ldots, m_k\}$, of the sequence of messages delivered by the Local Broadcast service in focal point h.

We say that a node i is *up-to-date* with respect to M if, after receiving message m_k, the following hold: (i) $data.tag_i$ is at least as large as the largest tag in any message in M, and $data.value_i$ is set to the value associated with that tag, (ii) for every $\langle confirm, oid, tag\rangle$ message in M, $tag \in data.confirmed_i$, (iii) $data.conf\text{-}id_i$ is at least as large as the largest configuration in any message in M, (iv) if there exists a message in M: $\langle recon\text{-}done, data.conf\text{-}id_i\rangle$, then $data.recon\text{-}ip_i$ is false, and (v) for all messages $\langle message, oid, payload\rangle \in M$, $oid \in complete\text{-}ops_i$. Essentially, a node is up-to-date with respect to a set of messages, M, if its $data$ and $complete\text{-}ops$ state is equivalent to the state of a node that has processed all the messages in M. Notice, then the following two facts:

Lemma C.1 *If i receives all the messages in M, and i is active on receiving all the messages in M, then i is up-to-date with respect to M.*

Lemma C.2 *If i is up-to-date with respect to M_1, and i is up-to-date with respect to M_2, then i is up-to-date with $M = M_1$ concatenated with M_2.*

The next lemma states that if two nodes both begin an execution in a given focal point, then as long as they remain in the focal point, they have the same state.

Lemma C.3 *Let i and j be two mobile hosts. Assume that node i and node j both begin execution α in focal point h, and both remain in focal point h until each receives message m_k. Then, immediately after the message m_k is received, the $data$ and $complete\text{-}ops$ state of i equals the same state of j.*

Proof. Both i and j receive every message in M. Both start in the same initial state, and handle the same requests in the same order. Therefore their required states are equal. □

The next lemma shows that if i enters focal point h and receives a join acknowledgment from a node that is up-to-date, then i is also up-to-date.

Lemma C.4 *Let i and j be two mobile hosts, and assume that at some point in α, i enters focal point h. Let m_a be the message in M corresponding with i's local broadcast of a* join-req.

Let $M' = \{m_0, \ldots, m_a\}$ be a prefix of M. Assume that node j is up-to-date with respect to M'. Also, assume that j sends a join-req-ack *message in response to i's joining.*

If i receives message m_k, and $status_i =$ active when i receives m_k, then i is up-to-date with respect to $\{m_0, \ldots, m_k\}$.

Proof. Let m_b by the join-req-ack message sent by j in response to i joining. Notice that message m_b contains the following information from M': (i) the maximum tag, and associated value, (ii) the set of all confirmed tags, (iii) the ID of the largest configuration and whether that reconfiguration is complete, the IDs of all operations (i.e., $\mathit{complete\text{-}ops}_j$). Therefore, when node i receives message m_b, its state accurately reflects having received messages $m_1, \ldots, m_a$, and i is up-to-date with respect to M'.

Next, notice that i has entered focal point h before sending message m_a, the join request. Therefore, by assumption of the Local Broadcast service, node i receives messages $m_{a+1}, \ldots, m_k$. Therefore, node i is up-to-date with respect to $m_{a+1}, \ldots, m_k$. By Lemma C.2, then, i is up-to-date with respect to M. $\square$

Lemma C.5 *Assume* $\mathit{status}_i = \mathsf{active}$ *when some event occurs in* α*. Then there exists a sequence of nodes* $i_0, \ldots, i_n$*, where* $i = i_n$*, such that the following hold: (i) node* i_0 *begins* α *in focal point* h*, (ii) for all* $k' < n$*, node* $i_{k'}$ *sends a* join-req-ack *in response to a* join-req *by node* $i_{k'+1}$*, and (iii) for all* $k' < n$*, node* $i_{k'+1}$ *receives the* join-req-ack *message sent by node* $i_{k'}$*.*

Proof. A node $i_{k'}$ only sets its status to active if (i) the node begins the execution in focal point h, or (ii) the node receives a join-req-ack, which must have been sent by some node $i_{k'-1}$ that had previously set its status to active. There are only a finite number of join-req-ack messages preceding the chosen event in α, and therefore some node, designated i_0, must have begun the execution in focal point h. $\square$

Finally, we show that the join protocol works:

Lemma C.6 *Let* i *be a mobile host. Assume that* i *is in* h *and receives message* m_k*. Also, assume that* $\mathit{status}_i = \mathsf{active}$ *when* m_k *is delivered. Then,* i *is up-to-date with respect to* M*.*

Proof. Let $i_0, \ldots, i_n$ be the sequence of mobile hosts such that $i = i_k$, and (i) node i_0 begins α in focal point h, (ii) for all $k' < n$, node $i_{k'}$ sent a join-req-ack in response to a join-req by node $i_{k'+1}$, and (iii) for all $k' < n$, node $i_{k'+1}$ receives the join-req-ack message sent by node $i_{k'}$. This sequence exists, as per Lemma C.5.

We show the result by induction on ℓ, with respect to the sequence $i_0, \ldots, i_n$. For all $0 \leq \ell < n$, let m_{i_ℓ} be the join-req-ack message sent from i_ℓ and received by $i_{\ell+1}$, and let m_{j_ℓ} be the $litjoin - req$ message sent by $i_{\ell+1}$. First, it is clear that i_0 is up-to-date with respect to $\{m_0, \ldots, m_{j_0}\}$, as it has received messages $\{m_0, \ldots, m_{j_0}\}$, and clearly remains alive and in h until sending $m_{i_0} > m_{j_0}$.

Next, consider i_ℓ, for some $\ell < n - 1$. Assume, inductively, that i_ℓ is up-to-date with respect to $\{m_0, \ldots, m_{j_\ell}\}$. Then, by Lemma C.4, $i_{\ell+1}$ is up-to-date with respect to $\{m_0, \ldots, m_{j_{\ell+1}}\}$ (applied where $i = i_{\ell+1}$, $j = i_\ell$, $a = j_\ell$, and $k = j_{\ell+1}$). Note that $i_{\ell+1}$ does not fail or leave h before receiving $m_{j_{\ell+1}}$, as by assumption it sends a response to this join request.

Therefore, i_{n-1} is up-to-date with respect to $\{m_0, \ldots, m_{j_{n-1}}\}$ – that is, the join-req message sent by i_n. Then, by another application of Lemma C.4, $i = i_n$ is up-to-date with respect to M (where $i = i_n$, $j = i_{n-1}$, $a = j_\ell$, and the k is the same). $\square$

Next, we show that every node in the focal point maintains the same replicated state:

Lemma C.7 *Assume that* i *and* j *are in focal point* h*, and both receive message* m_k*, and* i *and* j *are each active on receiving* m_k*. Then,* data_i *at the time when* i *receives* m_k *is equal to* data_j *at the time when* j *receives* m_k*. Similarly,* $\mathit{complete\text{-}ops}_i$ *at the time when* i *receives* m_k *is equal to* $\mathit{complete\text{-}ops}_j$ *at the time* j *receives* m_k*.*

Proof. By Lemma C.6, both i and j are up-to-date with respect to M. By definition of up-to-date, node i has tag at least as large as the largest tag in M, and the associated value. However, when i receives message m_k, i has only received Local Broadcast messages from some subset of M, so the tag of i can be no larger than the tags in M, so i has set $\mathit{data.tag}$ to the largest tag in M, and $\mathit{data.value}$ to the associated value. By a similar argument, $\mathit{data.conf\text{-}id}$ is set to the configuration ID of the largest configuration in M, and

$data.confirmed_i$ is exactly equal to the set of confirmed tags in M. Again by the definition of up-to-date, if M contains a recon-done message for $data.conf\text{-}id_i$, then $data.recon\text{-}ip_i = \mathsf{f}alse$. Since i receives no other messages aside from those in M, if there is no recon-done message in M for $data.conf\text{-}id_i$, then either $data.recon\text{-}ip_i = \mathsf{true}$ or $data.conf\text{-}id_i = \langle 0, 0, 0\rangle$. Similarly, $complete\text{-}ops_i$ is equal to all the operations in M.

The exact same logic holds for node j. Therefore the conclusion follows. □

Finally, we show that if two mobile hosts both send a response to a given request, then the response is the same. This allows us to focus on the actions of a single mobile host, rather than having to argue about every mobile host.

Theorem C.8 *Assume that at some point in the execution, node i has an item in geoc-queue$_i$ of the following form: $\langle op_1, oid, payload_1\rangle$. Further, assume that at some (other) point in the execution node j has an item in geoc-queue$_j$ of the following form: $\langle op_2, oid, payload_2\rangle$. Then, $op_1 = op_2$, and $payload_1 = payload_2$.*

Proof. First, both nodes i and j must be active, if they enqueued a GeoCast message. Next, notice that since both messages in the two GeoCast queues have the same oid, and therefore must have both been sent in response to the same Local Broadcast message, the first rebroadcast of the GeoCast request. Consider the message, m, received from the Local Broadcast service that caused i to enqueue the message on the GeoCast queue. Also, consider the message, m', that is ordered immediately before m by the local broadcast service. After receiving message m', nodes i and j have the same $data$ and $complete\text{-}ops$ state, by Lemma C.7. Therefore the message enqueued in response to message m will be the same in both cases. □

We next show that the Focal Point Emulator acts as in an atomic manner.

Theorem C.9 *Assume that node i, in focal point h, enqueues a response to an operation, π, in focal point h. Assume that after node i does the* geoc-send *of the response, node i' receives a* geoc-rcv(get, ...) *request. If some node j in focal point h eventually sends a response to the* get *request, then the tag for this response is no smaller than the tag for π, and the configuration ID for this request is no smaller than the configuration ID for π.*

Proof. Assume that i enqueues the response to π as a result of message m_k. Assume that j sends a response as a result of message m_ℓ. Since node j is active and receives m_ℓ, by Lemma C.6, j is up-to-date with respect to $\{m_0, \ldots, m_\ell\}$. Therefore, the tag of j is at least as large as the tag of m_k, since $m_k \in \{m_0, \ldots, m_\ell\}$. Also, by the same logic, the configuration ID of j is at least as large as the configuration ID of m_k. Therefore the conclusion follows. □

A final two lemmas handle the issue of the $confirmed$ flag and the $recon\text{-}ip$ flag.

Lemma C.10 *If node i ever sends a response including tag t where $confirmed$ is true, then some node j previously performed a* geoc-rcv(confirm, $*, t)_j$.

Proof. Let m_k be the Local Broadcast message that causes i to enqueue the response confirming tag t. By Lemma C.6, i is up-to-date with respect to $\{m_0, \ldots, m_k\}$. Therefore, the set of confirmed tags is exactly the set of tags confirmed by confirm messages in $\{m_0, \ldots, m_k\}$. □

Lemma C.11 *If node i ever sends a response including $recon\text{-}ip$ =* false *for some configuration $cid \neq \langle 0, 0, 0\rangle$, then some node j previously performed a* geoc-rcv(recon-done, $*, cid$).

Proof. Let m_k be the Local Broadcast message that causes i to enqueue the response indicating that reconfiguration cid is done. By Lemma C.6, i is up-to-date with respect to $\{m_0, \ldots, m_k\}$. Therefore, $data.recon\text{-}ip_i$ is only false as a result of a recon-done message in $\{m_0, \ldots, m_k\}$. □

C.2 Operation Manager

In this section, we show that the Operation Manager guarantees atomic consistency. In order to prove that all executions of the Operation Manager are atomic, we use the definition of atomicity based on four sufficient conditions. A memory is said to be *atomic*:

- If all the read and write operations that are invoked complete, then the read and write operations for object x can be partially ordered by an ordering $\prec$, so that: (i) No operation has infinitely many other operations ordered before it. (ii) The partial order is consistent with the external order of invocations and responses, that is, there do not exist read or write operations π_1 and π_2 such that π_1 completes before π_2 starts, yet $\pi_2 \prec \pi_1$. (iii) All write operations are totally ordered, and every read operation is ordered with respect to all the writes. (iv) Every read operation that is ordered after any writes returns the value of the last write preceding it in the partial order; any read operation ordered before all writes returns v_0.

This definition is equivalent to the other common definition: a distributed memory is atomic if it is, in some sense, equivalent to a serial, monolithic, memory. (See Lemma 13.16 in [19].) The goal is to show that these four conditions of atomicity hold. For a read or write operation, π, initiated at mobile host i, we define $tag(\pi)$ as follows: $tag(\pi) = op.tag_i$ at the instant when $\mathsf{read\text{-}ack}_i$ or $\mathsf{write\text{-}ack}_i$ occurs. We then define the partial order $\prec$: (a) for any two read or write operations π_1 and π_2, $tag(\pi_1) < tag(\pi_2) \implies \pi_1 \prec \pi_2$, (b) for any write operation π_1, and any read operation π_2, $tag(\pi_1) = tag(\pi_2) \implies \pi_1 \prec \pi_2$.

Condition (i) is immediate. To see Condition (iii), notice that every write operation has a unique tag determined by the global clock, with ties broken by processor id, implying that every write operation is totally ordered by $\prec$. Every read operation is ordered with respect to every write operation either by condition (a), if the tags of the two operations are unequal, or by condition (b), if the tags are equal. Condition (iv) is also straightforward: if a read operation has the same tag as a write operation, then it returns the value of that write operation; by condition (b) of $\prec$, a read operation comes after the write operation with the same tag.

The key condition, then, to guarantee atomic consistency is Condition (ii):

Theorem C.12 *If π_1 and π_2 are read or write operations, and π_1 completes before π_2 begins, then $tag(\pi_1) \leq tag(\pi_2)$, and if π_2 is a write operation then $tag(\pi_1) < tag(\pi_2)$.*

We first consider the case where π_2 is a write operation:

Lemma C.13 *If π_1 is a read or write operation, and π_2 is a write operation, and π_1 completes before π_2 begins, then $tag(\pi_1) < tag(\pi_2)$.*

Proof. The result follows immediately by the choice of $tag(\pi_2)$: $op.tag_i$ is defined in the $\mathsf{write}(v)_i$ action, and not modified until the $\mathsf{write\text{-}ack}_i$ occurs. It is chosen uniquely using the real-time clock, and thus must be larger than any prior operation that completes earlier. □

The key difficulty in the next part of the proof is related to reconfiguration: if no reconfiguration occurs, then the quorum intersection property guarantees that the operations are monotonic in their tags. We therefore need a few lemmas related to reconfiguration.

Lemma C.14 *Assume that at mobile host i, $recon\text{-}ip_i = \mathsf{false}$. Then either $conf\text{-}id_i = \langle 0, 0, 0\rangle$, or for some j a $\mathsf{recon\text{-}ack}(conf\text{-}id)_j$ occurs in α, no later than the event at which $recon\text{-}ip_i = \mathsf{false}$.*

Proof. There are three cases in which $recon\text{-}ip_i = \mathsf{false}$: (i) $conf\text{-}id = \langle 0, 0, 0\rangle$, (ii) i performs a $\mathsf{recon\text{-}ack}(conf\text{-}id)_i$, or (iii) i receives a GeoCast message from some focal point, say, h, indicating that reconfiguration of *conf-id* is complete. If Case 1 is true, then we are done. Similarly, if Case 2 is true, then $j = i$ and we are done.

Assume, by way of contradiction, that $\mathit{conf\text{-}id} \neq \langle 0, 0, 0 \rangle$, and that there does not exist j that performs a $\mathsf{recon\text{-}ack}(\mathit{conf\text{-}id})_j$ prior to the required event. Therefore, for every node i' such that $\mathit{recon\text{-}ip}_{i'} = \mathsf{false}$, Case 3 is true. If Case 3 is true for some node i', then by Lemma C.11, there exists a node, i'', that GeoCast focal point h a message indicating that the reconfiguration of *conf-id* is complete. However, by assumption, Case 3 is true for i'' too. This leads to an infinite sequence of nodes receiving notification that the reconfiguration of *conf-id* is complete. Yet there are only a finite number of messages GeoCast prior to the point in α where $\mathit{recon\text{-}ip}_i = \mathsf{false}$. Therefore, for some node j, Case 1 or Case 2 holds, and we are done. □

We say the π is a *one-phase* read operation if π is initiated by a $\mathsf{read}()_i$ action, is followed by a $\mathsf{read\text{-}ack}(val)_i$, and no $\mathsf{read\text{-}2}()_i$ occurs in between. A *two-phase* read operation is a read operation that is not a one-phase read. Also, for a read or write operation π, we define the *first-phase-end*(π) event as follows: (i) if π is a write operation, *first-phase-end*(π) is the write-ack event that completes π, (ii) if π is a one-phase read operation, *first-phase-end*(π) is the read-ack event that completes π, (iii) if π is a two-phase read operation, *first-phase-end*(π) is the read-2 event that concludes the first phase of π.

For a two-phase read or write operation π initiated at mobile host i, we define the $\mathit{conf}(\pi) = \mathit{op.conf\text{-}id}$ at the time when (i) if π is a write operation, the $\mathsf{write\text{-}ack}_i$ occurs, (ii) if π is a read operation, the $\mathsf{read\text{-}2}_i$ occurs.

First we consider the case where π_1 is a two-phase read operation. We begin by considering two operations, π_1 and π_2, that both complete using the same configuration; that is, no reconfiguration occurs during these operations.

Lemma C.15 *Assume operation π_1 is a two-phase read or write operation, and π_2 is a read operation. Assume that π_1 completes in α before π_2 begins, and that $\mathit{conf}(\pi_1) = \mathit{conf}(\pi_2)$. Then $\mathit{tag}(\pi_1) \leq \mathit{tag}(\pi_2)$.*

Proof. Assume that $\mathit{conf}(\pi_1) = \langle \mathit{cl}, \mathit{pid}, \mathit{cname} \rangle$. As part of operation π_1, either a write_i or a $\mathsf{read\text{-}2}_i$ occurs in α. Both of these actions initiate a *put* operation, GeoCasting the following message to various focal points: $\langle \mathsf{put}, \mathit{op.oid}_i, \mathit{op.tag}_i, \mathit{op.value}_i, \mathit{cf}, \mathit{conf\text{-}id}_i \rangle$. When the read-ack or write-ack occurs that concludes π_1, there exists a quorum of focal points, $p \in P_{\mathit{cname}}$, such that each focal point in p has GeoCast a response to the *put* operation.

Operation π_2 is initiated by a $\mathsf{read}()_j$ operation. This causes a *get* operation, GeoCasting the following message to various focal points: $\langle \mathsf{get}, \mathit{op.oid}_j, \mathit{conf\text{-}id}_j \rangle$. When the $\mathsf{read\text{-}2}_j$ or $\mathsf{read\text{-}ack}_j$ occurs that concludes the first phase of π_2, there exists a quorum of focal points, $g \in P_{\mathit{cname}}$, such that each focal point in g has GeoCast a response to the *get* operation.

By the intersection properties of quorums, there exists a focal point $h \in g \cap p$. By Theorem C.9, the response sent to j for the *get* operation must have a tag no smaller than the tag received by h from the preceding *put* operation. Therefore the desired inequality holds. □

Next we will show that if π_1 precedes π_2, then π_1 cannot complete in a larger configuration than π_2.

Lemma C.16 *Assume operation π_1 completes in α at node i before operation π_2 begins at node j. Assume that π_1 is a two phase read or write operation, and π_2 is a read operation. Then $\mathit{conf}(\pi_1) \leq \mathit{conf}(\pi_2)$.*

Proof. As part of operation π_1, either a write_i or a $\mathsf{read\text{-}2}_i$ occurs in α. Both of these actions initiate a *put* operation, GeoCasting the following message to various focal points: $\langle \mathsf{put}, \mathit{op.oid}_i, \mathit{op.tag}_i, \mathit{op.value}_i, \mathit{cf}, \mathit{conf\text{-}id}_i \rangle$. There are two cases to consider. First, assume that when the $\mathsf{read\text{-}ack}_i$ or $\mathsf{write\text{-}ack}_i$ occurs that concludes

π_1, *op.recon-in-progress*$_i$ = true. Then there exist quorums of focal points, $p_0 \in P_0$ and $p_1 \in P_1$, such that each focal point in $p_0 \cup p_1$ has GeoCast a response to the *put* operation.

Assume that $conf(\pi_2) = \langle cl, pid, cname\text{-}2\rangle$. Operation π_2 is initiated by a $\mathsf{read}()_j$ operation. This causes a *get* operation, GeoCasting the following message to various focal points: $\langle \mathsf{get}, op.oid_j, conf\text{-}id_j\rangle$. When the $\mathsf{first\text{-}phase\text{-}end}(\pi_2)_j$ occurs, there exists a quorum of focal points, $g \in G_{cname\text{-}2}$, such that each focal point in g has GeoCast a response to the *get* operation. By the intersection properties of quorums, there exists a focal point $h \in g \cap (p_0 \cup p_1)$. By Theorem C.9, the response sent to j for the *get* operation must have a configuration-id no smaller than the configuration-id received by h from the preceding *put* operation. Therefore $conf(\pi_1) \leq conf(\pi_2)$.

Assume, instead, that when the $\mathsf{read\text{-}ack}_i$ or $\mathsf{write\text{-}ack}_i$ occurs that concludes π_1, *op.recon-in-progress*$_i$ = false. Then, *recon-ip*$_i$ = false when $\mathsf{first\text{-}phase\text{-}end}(\pi_1)$ occurs. By Lemma C.14, either $conf\text{-}id_i = \langle 0,0,0\rangle$, or there exists some node i' that performs a $\mathsf{recon\text{-}ack}(conf\text{-}id_i)_{i'}$ prior to $\mathsf{first\text{-}phase\text{-}end}(\pi_1)$. If $conf\text{-}id_i = \langle 0,0,0\rangle$, then clearly $conf(\pi_1) \leq conf(\pi_2)$.

Otherwise, let i' be the mobile host that performs a a $\mathsf{recon\text{-}ack}(conf\text{-}id_i)_{i'}$ prior to $\mathsf{first\text{-}phase\text{-}end}(\pi_1)$. This means that a $\mathsf{recon\text{-}upgrade\text{-}2}(conf\text{-}id_i)_{i'}$ occurs in α prior to the end of the first phase of π_1. Then, there exist quorums of focal points, $p_0 \in P_0$ and $p_1 \in P_1$, such that each focal point in $p_0 \cup p_1$ has GeoCast a response to the recon's *get* operation. In particular, each of these focal points has been notified of $conf\text{-}id_i$.

When the $\mathsf{first\text{-}phase\text{-}end}(\pi_2)$ occurs, there exists a quorum of focal points, $g \in P_{cname\text{-}2}$, such that each focal point in g has GeoCast a response to π_2's *get* operation. By the intersection properties of quorums, there exists a focal point $h \in g \cap (p_0 \cup p_1)$. By Theorem C.9, the response sent to j for π_2's *get* operation must have a configuration-id no smaller than the configuration-id received by h from the preceding recon's *get* operation. Therefore $conf(\pi_1) \leq conf(\pi_2)$. □

The remaining lemma, then, addresses the case where $conf(\pi_1) < conf(\pi_2)$.

Lemma C.17 *Assume operation π_1 is a two-phase read or write operation, and completes at node i. Assume that π_2 is a read operation initiated at node j that completes, and that π_1 completes before π_2 begins. Additionally, assume that $conf(\pi_1) < conf(\pi_2)$. Then $tag(\pi_1) \leq tag(\pi_2)$.*

Proof. There are three cases to consider. First, assume that *op.recon-in-progress*$_j$ = true when the $\mathsf{read\text{-}2}_j$ or $\mathsf{read\text{-}ack}_j$ that concludes the first phase of π_2 occurs. During the first phase, messages of the following form are sent out to various focal points: $\langle \mathsf{get}, op.oid_j, conf\text{-}id_j\rangle$. Then, when the $\mathsf{read\text{-}2}_j$ or $\mathsf{read\text{-}ack}_j$ occurs, there exist quorums of focal points, $p_0 \in P_0$ and $p_1 \in P_1$, such that each focal point in $p_0 \cup p_1$ has GeoCast a response to the *get* operation.

Let $conf(\pi_1) = \langle cl, pid, cname\rangle$. As part of operation π_1, either a write_i or a $\mathsf{read\text{-}2}_i$ occurs in α. Both of these actions initiate a *put* operation, GeoCasting the following message to various focal points: $\langle \mathsf{put}, op.oid_i, op.tag_i, op.value_i, cf, conf\text{-}id_i\rangle$. Then there exists a quorum of focal points, $p \in P_{cname}$ such that each focal point in p_{cname} has GeoCast a response to the *put* operation. By the intersection properties of quorums, there exists a focal point $h \in g \cap (p_0 \cup p_1)$. By Theorem C.9, the response sent to j for the *get* operation must have a tag no smaller than the tag received by h from the preceding *put* operation. Therefore $tag(\pi_1) \leq tag(\pi_2)$.

The second case is the opposite: assume that *op.recon-in-progress*$_i$ = true when the $\mathsf{read\text{-}ack}$ or $\mathsf{write\text{-}ack}$ that concludes π_1 occurs. In this case, there exist quorums of focal points $g_0 \in G_0$ and $g_1 \in G_1$ such that every focal point in $g_0 \cup g_1$ sent a response to a *put* operation that included $tag(\pi_1)$. Also, there exists a quorum of focal points $p \in P_{cname\text{-}2}$, where $conf\text{-}id(\pi_2) = \langle cl, pid, cname\text{-}2\rangle$, such that every focal point in p has sent a response to the *get* operation initiated by π_2. There exists a focal point $h \in p \cap (g_0 \cup g_1)$, and it again follows from Theorem C.9 that $tag(\pi_1) \leq tag(\pi_2)$.

For the third case, assume that $op.recon\text{-}in\text{-}progress_i = \mathsf{false}$ and $op.recon\text{-}in\text{-}progress_j = \mathsf{false}$, when, respectively, the $\mathsf{read\text{-}ack}_i$ or $\mathsf{write\text{-}ack}_i$ of π_1 occur, and the $\mathsf{read}()_j$ of π_2 occur. Notice that $conf(\pi_2) \neq \langle 0,0,0\rangle$, since it is strictly larger than $conf(\pi_1)$. Then by Lemma C.14, there exists some mobile host r that performs a $\mathsf{recon\text{-}ack}(conf(\pi_2))_r$ prior to $\mathsf{read}()$ of π_2.

This means that a $\mathsf{recon\text{-}upgrade}(conf(\pi_2))_r$ occurs prior to the first phase of π_2. This initiates sending *get* messages to various focal points of the following form: $\langle \mathsf{get}, op.oid_r, conf\text{-}id_r\rangle$. When $\mathsf{recon\text{-}upgrade\text{-}2}(conf(\pi_2))_r$ occurs, there exist quorums of focal points, $g_0 \in G_0$ and $g_1 \in G_1$, such that each focal point in $g_0 \cup g_1$ has GeoCast a response to the *get* operation. When the $\mathsf{read\text{-}ack}_i$ or $\mathsf{write\text{-}ack}_i$ occurs that concludes the second phase of π_1, there exists a quorum of focal points, $p \in P_{cname}$, such that each focal point in p has GeoCast a response to the *put* operation initiated by the second phase of π_1. By the intersection properties of quorums, there exists a focal point $h \in p \cap (g_0 \cup g_1)$. If the *get* operation of the recon precedes the *put* of π_1 at focal point h, then by Theorem C.9, the response sent to i include configuration $conf(\pi_2)$, and $op.recon\text{-}in\text{-}progress_i$ would be set to true during the second phase of π_1. Instead, therefore, the *put* operation of π_1 must precede the *get* operation of the recon. By Theorem C.9, the response sent to r by focal point h for the *get* operation must have a tag no smaller than the tag received by h from the preceding *put* operation of π_1.

Now consider the second phase of the reconfiguration. During the second phase, *put* messages are sent to various focal point. When the $\mathsf{recon\text{-}ack}(conf(\pi_2))_r$ occurs, there exists a quorum of focal point $p' \in P_{cname\text{-}2}$ such that each focal point has responded to the *put* request. Consider again the first phase of π_2. When the first phase of π_2 completes with $\mathsf{first\text{-}phase\text{-}end}(\pi_2)$, there exists a quorum of focal points $g' \in G_{cname\text{-}2}$ such that every node in g' has sent a response to j for the *get* request. By quorum intersection, there exists $h' \in p' \cap h'$. And finally, by Theorem C.9, the response sent to j for the *get* operation must have a tag no smaller than the tag received by h' from the preceding *put* operation. Once again, $tag(\pi_1) \leq tag(\pi_2)$. □

Finally we combine these results to prove the main theorem.

Proof (Theorem C.12). First, assume that π_2 is a write operation. Then Lemma C.13 implies that $tag(\pi_1) < tag(\pi_2)$.

Next, assume that π_2 is a write operation and π_1 is a two phase read or a write operation. Lemma C.16 implies that $conf(\theta) \leq conf(\phi)$. Lemma C.15 and Lemma C.17 then imply that $tag(\pi_1) \leq tag(\pi_2)$.

Finally, assume that π_1 is a one-phase read operation and π_2 is a read operation. Assume π_1 completes at mobile host i. Define $tag(\pi_1) = op.tag_i$, at the time the matching $\mathsf{read\text{-}ack}_i$ occurs, which is consistent with our prior definitions. Since π_1 is a one-phase read operation, $tag(\pi_1) \in confirmed_i$ when the $\mathsf{read\text{-}ack}_i$ occurs. There are two ways in which a tag is added to confirmed: either i itself added the tag to $confirmed_i$ (by completing an earlier operation with tag equal to $tag(\pi_1)$), or mobile host i received a message from some focal point, h, indicating that the tag was confirmed. By Lemma C.10, focal point h would only send such a confirmation if it had previously received a message indicating that the tag was confirmed. Therefore, in either case, there exists some node i' that performs a $\mathsf{read\text{-}ack}$ or a $\mathsf{write\text{-}ack}$, and adds $tag(\pi_1)$ to $confirmed_{i'}$. Label this earlier operation pi'_1. Then, it is clear that $tag(\pi'_1) = tag(\pi_1)$. Also, π'_1 completes before π_1 begins, and therefore completes before π_2 begins. By the prior argument, then, $tag(\pi'_1) \leq tag(\pi_2)$, implying that $tag(\pi_1) \leq tag(\pi_2)$. □